Nginx 中文官方文档

wizardforcel

Published with GitBook



目錄

Nginx 中文官方文档

来源：www.nginx.cn

整理：飞龙

Nginx功能概述

HTTP基础功能：

- 处理静态文件，索引文件以及自动索引；
- 反向代理加速(无缓存)，简单的负载均衡和容错；
- FastCGI，简单的负载均衡和容错；
- 模块化的结构。过滤器包括gzipping, byte ranges, chunked responses, 以及 SSI-filter 。在SSI过滤器中，到同一个 proxy 或者 FastCGI 的多个子请求并发处理；
- SSL 和 TLS SNI 支持；

IMAP/POP3 代理服务功能：

- 使用外部 HTTP 认证服务器重定向用户到 IMAP/POP3 后端；
- 使用外部 HTTP 认证服务器认证用户后连接重定向到内部的 SMTP 后端；
- 认证方法：
- POP3: POP3 USER/PASS, APOP, AUTH LOGIN PLAIN CRAM-MD5;
- IMAP: IMAP LOGIN;
- SMTP: AUTH LOGIN PLAIN CRAM-MD5;
- SSL 支持；
- 在 IMAP 和 POP3 模式下的 STARTTLS 和 STLS 支持；

支持的操作系统：

- FreeBSD 3.x, 4.x, 5.x, 6.x i386; FreeBSD 5.x, 6.x amd64;
- Linux 2.2, 2.4, 2.6 i386; Linux 2.6 amd64;
- Solaris 8 i386; Solaris 9 i386 and sun4u; Solaris 10 i386;
- MacOS X (10.4) PPC;

结构与扩展：

- 一个主进程和多个工作进程。工作进程是单线程的，且不需要特殊授权即可运行；
- kqueue (FreeBSD 4.1+), epoll (Linux 2.6+), rt signals (Linux 2.2.19+), /dev/poll (Solaris 7 11/99+), select, 以及 poll 支持；
- kqueue支持的不同功能包括 EV_CLEAR, EV_DISABLE （临时禁止事件）, NOTE_LOWAT, EV_EOF, 有效数据的数目，错误代码；
- sendfile (FreeBSD 3.1+), sendfile (Linux 2.2+), sendfile64 (Linux 2.4.21+), 和 sendfilev (Solaris 8 7/01+) 支持；
- 输入过滤 (FreeBSD 4.1+) 以及 TCP_DEFER_ACCEPT (Linux 2.4+) 支持；
- 10,000 非活动的 HTTP keep-alive 连接仅需要 2.5M 内存。
- 最小化的数据拷贝操作；

其他**HTTP**功能：

- 基于IP 和名称的虚拟主机服务；
- Memcached 的 GET 接口；
- 支持 keep-alive 和管道连接；
- 灵活简单的配置；
- 重新配置和在线升级而无须中断客户的工作进程；
- 可定制的访问日志，日志写入缓存，以及快捷的日志回卷；
- 4xx-5xx 错误代码重定向；
- 基于 PCRE 的 rewrite 重写模块；
- 基于客户端 IP 地址和 HTTP 基本认证的访问控制；
- PUT, DELETE, 和 MKCOL 方法；
- 支持 FLV （Flash 视频）；
- 带宽限制；

实验特性：

- 内嵌的 `perl`
- 通过 `aio_read() / aio_write()` 的套接字工作的实验模块，仅在 FreeBSD 下。
- 对线程的实验化支持，FreeBSD 4.x 的实现基于 rfork()

Nginx 主要的英语站点是 http://sysoev.ru/en/

英语文档草稿由 Aleksandar Lazic 完成 点击 。

为什么选择Nginx

Nginx 是一个高性能的 Web 和反向代理服务器, 它具有有很多非常优越的特性:

作为 **Web** 服务器：相比 Apache，Nginx 使用更少的资源，支持更多的并发连接，体现更高的效率，这点使 Nginx 尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应，感谢 Nginx 为我们选择了 epoll and kqueue 作为开发模型.

作为负载均衡服务器：Nginx 既可以在内部直接支持 Rails 和 PHP，也可以支持作为 HTTP代理服务器 对外进行服务。Nginx 用 C 编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好的多。

作为邮件代理服务器: Nginx 同时也是一个非常优秀的邮件代理服务器（最早开发这个产品的目的之一也是作为邮件代理服务器），Last.fm 描述了成功并且美妙的使用经验。

Nginx 安装非常的简单，配置文件 非常简洁（还能够支持**perl**语法），**Bugs**非常少的服务器: Nginx 启动特别容易，并且几乎可以做到7*24不间断运行，即使运行数个月也不需要重新启动。你还能够在 不间断服务的情况下进行软件版本的升级。

Nginx安装

nginx可以使用各平台的默认包来安装，本文是介绍使用源码编译安装，包括具体的编译参数信息。

正式开始前，编译环境gcc g++ 开发库之类的需要提前装好，这里默认你已经装好。

ububtu平台编译环境可以使用以下指令

```
apt-get install build-essential
apt-get install libtool
```

centos平台编译环境使用如下指令

安装make：

```
yum -y install gcc automake autoconf libtool make
```

安装g++:

```
yum install gcc gcc-c++
```

下面正式开始

一般我们都需要先装pcre, zlib，前者为了重写rewrite，后者为了gzip压缩。
1.选定源码目录
可以是任何目录，本文选定的是/usr/local/src

```
cd /usr/local/src
```

2.安装**PCRE**库
ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/ 下载最新的 PCRE 源码包，使用下面命令下载编译和安装 PCRE 包：

```
cd /usr/local/src
wget ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/pcre-8.34.tar.gz
tar -zxvf pcre-8.34.tar.gz
cd pcre-8.34
./configure
make
make install
```

3.安装zlib库

http://zlib.net/zlib-1.2.8.tar.gz 下载最新的 zlib 源码包，使用下面命令下载编译和安装 zlib包：

```
cd /usr/local/src

wget http://zlib.net/zlib-1.2.8.tar.gz
tar -zxvf zlib-1.2.8.tar.gz
cd zlib-1.2.8
./configure
make
make install
```

4.安装ssl（某些vps默认没装ssl)

```
cd /usr/local/src
wget http://www.openssl.org/source/openssl-1.0.1c.tar.gz
tar -zxvf openssl-1.0.1c.tar.gz
```

5.安装nginx

Nginx 一般有两个版本，分别是稳定版和开发版，您可以根据您的目的来选择这两个版本的其中一个，下面是把 Nginx 安装到 /usr/local/nginx 目录下的详细步骤：

```
cd /usr/local/src
wget http://nginx.org/download/nginx-1.4.2.tar.gz
tar -zxvf nginx-1.4.2.tar.gz
cd nginx-1.4.2

./configure --sbin-path=/usr/local/nginx/nginx \
--conf-path=/usr/local/nginx/nginx.conf \
--pid-path=/usr/local/nginx/nginx.pid \
--with-http_ssl_module \
--with-pcre=/usr/local/src/pcre-8.34 \
--with-zlib=/usr/local/src/zlib-1.2.8 \
--with-openssl=/usr/local/src/openssl-1.0.1c

make
make install
```

--with-pcre=/usr/src/pcre-8.34 指的是pcre-8.34 的源码路径。

--with-zlib=/usr/src/zlib-1.2.7 指的是zlib-1.2.7 的源码路径。

安装成功后 /usr/local/nginx 目录下如下

```
fastcgi.conf            koi-win             nginx.conf.default
fastcgi.conf.default    logs                scgi_params
fastcgi_params          mime.types          scgi_params.default
fastcgi_params.default  mime.types.default  uwsgi_params
html                    nginx               uwsgi_params.default
koi-utf                 nginx.conf          win-utf
```

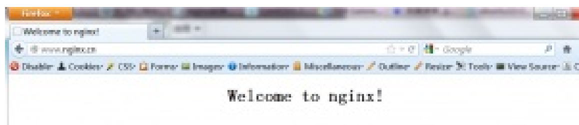
6.启动
确保系统的 80 端口没被其他程序占用，运行/usr/local/nginx/nginx 命令来启动 Nginx，

```
netstat -ano|grep 80
```

如果查不到结果后执行，有结果则忽略此步骤（ubuntu下必须用sudo启动，不然只能在前台运行）

```
sudo /usr/local/nginx/nginx
```

打开浏览器访问此机器的 IP，如果浏览器出现 Welcome to nginx! 则表示 Nginx 已经安装并运行成功。



到这里nginx就安装完成了，如果只是处理静态html就不用继续安装了

如果你需要处理php脚本的话，还需要安装php-fpm。

下面安装排错

附：可能遇到的错误和一些帮助信息

1.1编译pcre错误

```
libtool: compile: unrecognized option `-DHAVE_CONFIG_H'
libtool: compile: Try `libtool --help' for more information.
make[1]: *** [pcrecpp.lo] Error 1
make[1]: Leaving directory `/usr/local/src/pcre-8.34'
make: *** [all] Error 2
```

解决办法：安装g++,别忘了重新configure

```
apt-get install g++
apt-get install build-essential
make clean
./configure
make
```

1.2 make出错

```
make: *** No rule to make target `build', needed by `default'.  Stop.
./configure: error: SSL modules require the OpenSSL library.
You can either do not enable the modules, or install the OpenSSL library
into the system, or build the OpenSSL library statically from the source
with nginx by using --with-openssl=
```

```
option.
```

按照第4步的安装方法或
ubuntu下

```
apt-get install openssl
apt-get install libssl-dev
```

centos下

```
yum -y install openssl openssl-devel
```

2.nginx编译选项

make是用来编译的，它从Makefile中读取指令，然后编译。

make install是用来安装的，它也从Makefile中读取指令，安装到指定的位置。

configure命令是用来检测你的安装平台的目标特征的。它定义了系统的各个方面，包括nginx的被允许使用的连接处理的方法，比如它会检测你是不是有CC或GCC，并不是需要CC或GCC，它是个shell脚本，执行结束时，它会创建一个Makefile文件。nginx的configure命令支持以下参数：

- `--prefix=` *path* 定义一个目录，存放服务器上的文件，也就是nginx的安装目录。默认使用 `/usr/local/nginx。`
- `--sbin-path=` *path* 设置nginx的可执行文件的路径，默认为 `` `*prefix* `` /sbin/nginx`.
- `--conf-path=` *path* 设置在nginx.conf配置文件的路径。nginx允许使用不同的配置文件启动，通过命令行中的-c选项。默认为 `` `*prefix* `` /conf/nginx.conf`.
- `--pid-path=` *path* 设置nginx.pid文件，将存储的主进程的进程号。安装完成后，可以随时改变的文件名，在nginx.conf配置文件中使用 PID指令。默认情况下，文件名为 `` `*prefix* `` /logs/nginx.pid`.
- `--error-log-path=` *path* 设置主错误，警告，和诊断文件的名称。安装完成后，可以随时改变的文件名，在nginx.conf配置文件中 使用 的error_log指令。默认情况下，文件名为 `` `*prefix* `` /logs/error.log`.
- `--http-log-path=` *path* 设置主请求的HTTP服务器的日志文件的名称。安装完成后，可以随时改变的文件名，在nginx.conf配置文件中 使用 的access_log指令。默认情况下，文件名 为 `` `*prefix* `` /logs/access.log`.
- `--user=` *name* 设置nginx工作进程的用户。安装完成后，可以随时更改的名称在nginx.conf配置文件中 使用的 user指令。默认的用户名是nobody。
- `--group=` *name* 设置nginx工作进程的用户组。安装完成后，可以随时更改的名称在nginx.conf配置文件中 使用的 user指令。默认的为非特权用户。
- `` --with-select_module``--without-select_module 启用或禁用构建一个模块来允许服务器使用select() ``
- `` --with-poll_module``--without-poll_module `` 启用或禁用构建一个模块来允许服务器使用poll()方法。该模块将自动建立，如果平台不支持的kqueue，epoll，rtsig或/dev/poll。
- `--without-http_gzip_module` — 不编译压缩的HTTP服务器的响应模块。编译并运行此模块需要zlib库。
- `--without-http_rewrite_module` 不编译重写模块。编译并运行此模块需要PCRE库支持。
- `--without-http_proxy_module` — 不编译http_proxy模块。
- `--with-http_ssl_module` — 使用https协议模块。默认情况下，该模块没有被构建。建立

并运行此模块的OpenSSL库是必需的。

- `--with-pcre=` *path* — 设置PCRE库的源码路径。PCRE库的源码（版本4.4 - 8.30）需要从PCRE网站下载并解压。其余的工作是Nginx的./ configure和make来完成。正则表达式使用在location指令和 ngx_http_rewrite_module 模块中。
- `--with-pcre-jit` —编译PCRE包含“just-in-time compilation”（1.1.12中， pcre_jit指令）。
- `--with-zlib=` *path* —设置的zlib库的源码路径。要下载从 zlib（版本1.1.3 - 1.2.5）的并解压。其余的工作是Nginx的./ configure和make完成。ngx_http_gzip_module模块需要使用zlib 。
- `--with-cc-opt=` *parameters* — 设置额外的参数将被添加到CFLAGS变量。例如,当你在FreeBSD上使用PCRE库时需要使用: `--with-cc-opt="-I /usr/local/include。` .如需要需要增加 `select()支持的文件数量` : `--with-cc-opt="-D FD_SETSIZE=2048".`
- `--with-ld-opt=` *parameters* —设置附加的参数，将用于在链接期间。例如，当在FreeBSD下使用该系统的PCRE库,应指定: `--with-ld-opt="-L /usr/local/lib".`

典型实例(下面为了展示需要写在多行，执行时内容需要在同一行)

```
./configure
    --sbin-path=/usr/local/nginx/nginx
    --conf-path=/usr/local/nginx/nginx.conf
    --pid-path=/usr/local/nginx/nginx.pid
    --with-http_ssl_module
    --with-pcre=../pcre-4.4
    --with-zlib=../zlib-1.1.3
```

运行和控制Nginx

nginx命令行参数

不像许多其他软件系统，Nginx 仅有几个命令行参数，完全通过配置文件来配置

-c </path/to/config> 为 **Nginx** 指定一个配置文件，来代替缺省的。

-t 不运行，而仅仅测试配置文件。nginx 将检查配置文件的语法的正确性，并尝试打开配置文件中所引用到的文件。

-v 显示 nginx 的版本。

-V 显示 nginx 的版本，编译器版本和配置参数。

nginx控制信号

可以使用信号系统来控制主进程。默认，nginx 将其主进程的 pid 写入到 /usr/local/nginx/nginx.pid 文件中。通过传递参数给 ./configure 或使用 **pid** 指令，来改变该文件的位置。

主进程可以处理以下的信号：

TERM, INT	快速关闭
QUIT	从容关闭
HUP	重载配置 用新的配置开始新的工作进程 从容关闭旧的工作进程
USR1	重新打开日志文件
USR2	平滑升级可执行程序。
WINCH	从容关闭工作进程

尽管你不必自己操作工作进程，但是，它们也支持一些信号：

TERM, INT	快速关闭
QUIT	从容关闭
USR1	重新打开日志文件

nginx 启动、停止、重启命令

nginx启动

sudo /usr/local/nginx/nginx (nginx二进制文件绝对路径，可以根据自己安装路径实际决定)

nginx从容停止命令，等所有请求结束后关闭服务

ps -ef |grep nginx

kill -QUIT nginx主进程号

nginx 快速停止命令，立刻关闭**nginx**进程

ps -ef |grep nginx

kill -TERM nginx主进程号

如果以上命令不管用，可以强制停止

kill -9 nginx主进程号

如果嫌麻烦可以不用查看进程号，直接使用命令进行操作
其中/usr/local/nginx/nginx.pid 为nginx.conf中pid命令设置的参数，用来存放nginx主进程号的文件
kill -信号类型(HUP|TERM|QUIT) `cat /usr/local/nginx/nginx.pid`
例如

```
kill -QUIT `cat /usr/local/nginx/nginx.pid`
```

nginx重启命令

nginx重启可以分成几种类型

1.简单型，先关闭进程，修改你的配置后，重启进程。
kill -QUIT `cat /usr/local/nginx/nginx.pid`
sudo /usr/local/nginx/nginx
2.重新加载配置文件，不重启进程，不会停止处理请求
3.平滑更新nginx二进制，不会停止处理请求

使用信号加载新的配置

Nginx 支持几个信号，能在它运行时控制其操作。其中最普通的是 15 ，用来中止运行的进程：

```
# <strong>ps aux | egrep '(PID|nginx)'</strong>
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2213  0.0  0.0   6784  2036 ?        Ss   03:01   0:00 nginx: master process /u
# <strong>kill -15 2213</strong>
```

而最有趣的是能平滑改变 nginx 配置的选项（请注意，在重载前，要先测试一下配置文件）：

```
#<strong> nginx -t -c /etc/nginx/nginx.conf</strong>
2006/09/16 13:07:10 [info] 15686#0: the configuration file /etc/nginx/nginx.conf syntax i
2006/09/16 13:07:10 [info] 15686#0: the configuration file /etc/nginx/nginx.conf was test
#<strong> ps aux | egrep '(PID|nginx)'</strong>
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2213  0.0  0.0   6784  2036 ?        Ss   03:01   0:00 nginx: master process /u
<strong># kill -HUP 2213</strong>
```

当 nginx 接收到 HUP 信号，它会尝试先解析配置文件（如果指定配置文件，就使用指定的，否则使用默认的），成功的话，就应用新的配置文件（例如：重新打开日志文件或监听的套接 字）。之后，nginx 运行新的工作进程并从容关闭旧的工作进程。通知工作进程关闭监听套接字但是继续为当前连接的客户提供服务。所有客户端的服务完成后，旧的工作进程被关闭。 如果新的配置文件应用失败，nginx 将继续使用旧的配置进行工作。

平滑升级到新的二进制代码

你可以在不中断服务的情况下 - 新的请求也不会丢失，使用新的 nginx 可执行程序替换旧的（当升级新版本或添加/删除服务器模块时）。

首先，使用新的可执行程序替换旧的（最好做好备份），然后，发送 USR2 (kill -USR2 pid) 信号给主进程。主进程将重命名它的 **.pid** 文件为 **.oldbin** (比如：**/usr/local/nginx/logs/nginx.pid.oldbin**)，然后执行新的可执行程序，依次启动新的主进程和新的工作进程：

```
  PID  PPID USER    %CPU   VSZ WCHAN  COMMAND
33126     1 root     0.0  1164 pause  nginx: master process /usr/local/nginx/sbin/nginx
33134 33126 nobody   0.0  1368 kqread nginx: worker process (nginx)
33135 33126 nobody   0.0  1380 kqread nginx: worker process (nginx)
33136 33126 nobody   0.0  1368 kqread nginx: worker process (nginx)
36264 33126 root     0.0  1148 pause  nginx: master process /usr/local/nginx/sbin/nginx
36265 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36266 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36267 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
```

在这时，两个 nginx 实例会同时运行，一起处理输入的请求。要逐步停止旧的实例，你必须发送 WINCH 信号给旧的主进程，然后，它的工作进程就将开始从容关闭：

```
  PID  PPID USER    %CPU   VSZ WCHAN  COMMAND
33126     1 root     0.0  1164 pause  nginx: master process /usr/local/nginx/sbin/nginx
33135 33126 nobody   0.0  1380 kqread nginx: worker process is shutting down (nginx)
36264 33126 root     0.0  1148 pause  nginx: master process /usr/local/nginx/sbin/nginx
36265 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36266 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36267 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
```

一段时间后，旧的工作进程处理了所有已连接的请求后退出，就仅由新的工作进程来处理输入的请求了：

```
  PID  PPID USER    %CPU   VSZ WCHAN  COMMAND
33126     1 root     0.0  1164 pause  nginx: master process /usr/local/nginx/sbin/nginx
36264 33126 root     0.0  1148 pause  nginx: master process /usr/local/nginx/sbin/nginx
36265 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36266 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
36267 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
```

这时，因为旧的服务器还尚未关闭它监听的套接字，所以，通过下面的几步，你仍可以恢复旧的服务器：

- 发送 HUP 信号给旧的主进程 - 它将在不重载配置文件的情况下启动它的工作进程
- 发送 QUIT 信号给新的主进程，要求其从容关闭其工作进程
- 发送 TERM 信号给新的主进程，迫使其退出
- 如果因为某些原因新的工作进程不能退出，向其发送 KILL 信号

新的主进程退出后，旧的主进程会由移除 **.oldbin** 前缀，恢复为它的 **.pid** 文件，这样，一切就都恢复到升级之前了。

如果尝试升级成功，而你也希望保留新的服务器时，发送 QUIT 信号给旧的主进程使其退出而只留下新的服务器运行：

```
    PID  PPID USER    %CPU   VSZ WCHAN  COMMAND
  36264     1 root     0.0  1148 pause  nginx: master process /usr/local/nginx/sbin/ngi
  36265 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
  36266 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
  36267 36264 nobody   0.0  1364 kqread nginx: worker process (nginx)
```

配置符号参考

容量符号缩写

k,K	千字节
m,M	兆字节

例如, "8k", "1m" 代表字节数计量.

时间符号缩写

ms	毫秒
s	秒
m	分钟
h	小时
d	日
w	周
M	一个月, 30天
y	年, 365 天

例如, "1h 30m", "1y 6M". 代表 "1小时 30分", "1年零6个月".

优化 Nginx

Nginx使用hash表来协助完成请求的快速处理。

考虑到保存键及其值的hash表存储单元的大小不至于超出设定参数(hash bucket size)， 在启动和每次重新配置时，Nginx为hash表选择尽可能小的尺寸。

直到hash表超过参数(hash max size)的大小才重新进行选择. 对于大多数hash表都有指令来修改这些参数。例如，保存服务器名字的hash表是由指令 `server_names_hash_max_size` 和 `server_names_hash_bucket_size` 所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键值。因此，如果Nginx给出需要增大 hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.

事件模型

Nginx支持如下处理连接的方法（I/O复用方法）， 这些方法可以通过 `use` 指令指定。

- **select** - 标准方法。 如果当前平台没有更有效的方法，它是编译时默认的方法。你可以使用配置参数 `--with-select_module` 和 `--without-select_module` 来启用或禁用这个模块。
- **poll** - 标准方法。 如果当前平台没有更有效的方法，它是编译时默认的方法。你可以使用配置参数 `--with-poll_module` 和 `--without-poll_module` 来启用或禁用这个模块。
- **kqueue** - 高效的方法，使用于 FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X. 使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。
- **epoll** - 高效的方法，使用于Linux内核2.6版本及以后的系统。在某些发行版本中，如SuSE 8.2, 有让2.4版本的内核支持epoll的补丁。
- **rtsig** - 可执行的实时信号，使用于Linux内核版本2.2.19以后的系统。默认情况下整个系统中不能出现大于1024个POSIX实时(排队)信号。这种情况对于高负载的服务器来说是低效的；所以有必要通过调节内核参数 `/proc/sys/kernel/rtsig-max` 来增加队列的大小。可是从Linux内核版本2.6.6-mm2开始， 这个参数就不再使用了，并且对于每个进程有一个独立的信号队列，这个队列的大小可以用 RLIMIT_SIGPENDING 参数调节。当这个队列过于拥塞，nginx就放弃它并且开始使用 `poll` 方法来处理连接直到恢复正常。
- **/dev/poll** - 高效的方法，使用于 Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+.
- **eventport** - 高效的方法，使用于 Solaris 10. 为了防止出现内核崩溃的问题， 有必要安装 这个 安全补丁。

参考

Hash表原始文档

常见问题(FAQ)

- [#notwork 某些东东不工作 (URL重写, 代理, 路径, ...)]
- [#other 有没有其它类似的Web服务器]
- [#chroot 对于chroot的支持是否在计划之中?]
- [#usecase 在什么情况下使用Nginx比使用squid要好?]
- [#imapexample 有没有人能给出一个完整的.conf配置文件来详细的解读一下怎么配置和测试 IMAP 模块， 而不只是关于 IMAP 的只言片语啊?]
- [#smtpexample 怎么让Nginx成为以postfix做为后端的SMTP代理?]
- [#loadbalancing Nginx使用什么算法来实现负载均衡? 它能实现基于连接数的负载均衡吗?]
- [#proxy_buffering 我能关闭从代理服务器到后端服务器的缓存吗或者使用上传进度特性?]

某些东东不工作 (URL重写, 代理, 路径, ...)

例如： 如URL重写(rewrite)不工作了或者是unix的路径(/$PATH)的问题云云...

请仔细阅读 [NginxDebugging] 并且 逐行 查看错误日志。
如果你没找到错误 打起精神 试着到IRC或邮件列表里说明一下你碰到的问题。

有没有其它类似的Web服务器

- Cherokee
- Lighttpd (Lighty)
- thttpd

关于各自的优缺点请使用自己喜欢的搜索引擎查找 ;-)

对于chroot的支持是否在计划之中?

有人知道吗？

在什么情况下使用Nginx比使用squid要好? 反之亦然。

大体上来说nginx主要用于反向加速代理而不是像squid那样做为常规代理服务器。Nginx的最大优势在于高负载情况下内存和CPU的低消耗。 我不认为squid能给你带来比nginx更好的性能。

怎么让Nginx成为以postfix做为后端的SMTP代理?

有人知道不？

Nginx使用什么算法来实现负载均衡? 它能实现基于连接数的负载均衡吗?

目前Nginx使用简单的轮巡算法，所以无法做基本链接计数的负载均衡。 这个可能会在将来的版本中有所改变。

> 我能关闭从代理服务器到后端服务器的缓存吗或者使用上传进度特性?

基于 太多人询问下面的问题:

- 我能为了得到上传进度而关闭代理的缓存吗
- 使用nginx我怎么才能给用户显示上传进度
- ...

到目前为止 (2007-Apr-26) 还没有办法关闭到后端服务器的缓存.

调试 nginx

Nginx的一个 杀手级特性 就是你能使用 debug_connection 指令只调试 某些 连接。

这个设置只有是你使用 *--with-debug* 编译的nginx才有效。

鏍稿績妯″潡

Nginx主模块

这里是控制 Nginx 的基本功能的指令.

指令

- [#daemon daemon]
- [#debug_points debug_points]
- [#error_log error_log]
- [#include include]
- [#lock_file lock_file]
- [#master_process master_process]
- [#pid pid]
- [#ssl_engine ssl_engine]
- [#timer_resolution timer_resolution]
- [#user user group]
- [#worker_cpu_affinity worker_cpu_affinity]
- [#worker_priority worker_priority]
- [#worker_processes worker_processes]
- [#worker_rlimit_core worker_rlimit_core]
- [#worker_rlimit_nofile worker_rlimit_nofile]
- [#worker_rlimit_sigpending worker_rlimit_sigpending]
- [#working_directory working_directory]

daemon

语法:*daemon on | off*

缺省值:*on*

```
daemon  off;
```

Do not use the "daemon" and "master_process" directives in a production mode, these options are mainly used for development only. You can use `daemon off` safely in production mode with runit / daemontools however you can't do a graceful upgrade. `master_process off` should never be used in production.

生产环境中不要使用"daemon"和"master_process"指令，这些选项仅用于开发调试。

debug_points

语法:*debug_points [stop | abort]*

缺省值:*none*

```
debug_points stop;
```

There are some assertion points inside nginx that allow to stop nginx to attach the debugger, or to abort and to create the core file.

应该适用于调试，在调试器内设置断点之类的。

error_log

语法:*error_log file [debug | info | notice | warn | error | crit]*

缺省值:*${prefix}/logs/error.log*

Nginx 添加 `--with-debug 编译参数`, 你还能够使用以下配置:

```
error_log LOGFILE [ debug_core | debug_alloc | debug_mutex | debug_event
]: | debug_http | debug_imap ;
```

include

语法:*include file | **

缺省值:*none*

你可以在任意地方使用include指令实现配置文件的包含，类似于apache中的include方法，可减少主配置文件d。

`include` 指令还支持像下面配置一样的全局包含的方法，例如包含一个目录下所有以".conf"结尾的文件:

```
include vhosts/*.conf;
```

注意路径受到configure编译参数--prefix=<路径>指令的影响，如果没有指定，Nginx默认是被编译在/usr/local/nginx。

语法:*lock_file file*

缺省值:*compile-time option*

```
lock_file  /var/log/lock_file;
```

nginx uses accept mutex to serialize accept() syscalls. If nginx is built by gcc, Intel C++, or SunPro C++ compilers on i386, amd64, sparc64, and ppc64, then nginx uses the atomic instructions to implement the mutex. In other cases the lock file would be used.

master_process

语法:*master_process on | off*

缺省值:*on*

```
master_process  off;
```

Do not use the "daemon" and "master_process" directives in a production mode, these options are mainly used for development only.

生产环境中不要使用"daemon"和"master_process"指令，这些选项仅用于开发调试。

pid

语法:*pid file*

缺省值:*compile-time option* Example:

```
pid /var/log/nginx.pid;
```

进程id存储文件。可以使用 kill -HUP `cat /var/log/nginx.pid\` 对Nginx进行配置文件重新加载。

ssl_engine

语法:*ssl_engine engine*

缺省值:*system dependent*

Here you can set your preferred openssl engine if any available. You can figure out which one do you have with the commandline tool:

该指令用于指定openssl使用的引擎。你可以通过下面的命令行获知系统目前支持的openssl引擎

`openssl engine -t`

例如:

```
$ openssl engine -t
(cryptodev) BSD cryptodev engine
: [ available ]
(dynamic) Dynamic engine loading support
: [ unavailable ]
```

timer_resolution

语法:*timer_resolution t*

缺省值:*none*

Example:

```
timer_resolution  100ms;
```

The directive allows to decrease number gettimeofday() syscalls. By default gettimeofday() is called after each return from kevent(), epoll, /dev/poll, select(), poll().

But if you need an exact time in logs when logging $upstream_response_time, or $msec variables, then you should use `timer_resolution`.

user

语法:*user user [group]*

缺省值:*nobody nobody*

指定Nginx Worker进程运行用户，默认是nobody帐号。

例如:

```
user www users;
```

worker_cpu_affinity

语法:*worker_cpu_affinity cpumask [cpumask...]*

缺省值:*none*

Linux only.

With this option you can bind the worker process to a CPU, it calls sched_setaffinity().

仅适用于linux，使用该选项可以绑定worker进程和CPU.

For example,

```
worker_proceses     4;
worker_cpu_affinity 0001 0010 0100 1000;
```

Bind each worker process to one CPU only.

分别给每个worker进程绑定一个CPU.

```
worker_proceses     2;
worker_cpu_affinity 0101 1010;
```

Bind the first worker to CPU0/CPU2, bind the second worker to CPU1/CPU3. This is suitable for HTT.

将CPU0/CPU2绑定给第一个worker进程，将CPU1/CPU3绑定给第二个worker进程。

worker_priority

语法:*worker_priority [-] number*

缺省值:*on*

With this option you can give to all worker processes the priority (nice) you need/wish, it calls setpriority().

使用该选项可以给所有的worker进程分配优先值。

worker_processes

语法:*worker_processes number*

缺省值:*1*

e.g.:

```
worker_processes 5;
```

nginx has the ability to use more than one worker process for several reasons:

nginx可以使用多个worker进程，原因如下：

1. to use SMP
2. to decrease latency when workers blockend on disk I/O
3. to limit number of connections per process when select()/poll() is used

The `worker_processes` and `worker_connections` from the event sections allows you to calculate `maxclients` value: k

max_clients = worker_processes * worker_connections

worker_rlimit_core

语法:*worker_rlimit_core size*

缺省值: '

Maximum size of core file per worker;

worker_rlimit_nofile

语法: ***worker_rlimit_nofile limit*** 缺省值: **'**

Specifies the value for maximum file descriptors that can be opened by this process.

指定

worker_rlimit_sigpending

语法:*worker_rlimit_sigpending limit*缺省值: '

(Since Linux 2.6.8) Specifies the limit on the number of signals that may be queued for the real user ID of the calling process.

working_directory

语法: ***working_directory path*** 缺省值: ***--prefix***

This is the working directory for the workers. It's used for core files only. nginx uses absolute paths only, all relative paths in configuration files are relative to `--prefix==PATH`

Nginx事件模块

accept_mutex

Syntax:*accept_mutex [on | off]*

Default:*on*

nginx 使用连接互斥锁进行顺序的accept()系统调用.

accept_mutex_delay

Syntax:*accept_mutex_delay Nms;*

Default:*500ms*

如果一个进程没有互斥锁，它将延迟至少多长时间。默认情况下，延迟是500ms 。

debug_connection

Syntax:*debug_connection [ip | CIDR]*

Default:*none*

Since 0.3.54 this option support CIDR address format

This option gives you the ability to write debug log only for the clients of this IP/NET.

Several different directives are possible.

Example:

```
error_log /var/log/nginx/errors;
events {
  debug_connection   192.168.1.1;

}
```

devpoll_changes

devpoll_events

kqueue_events

epoll_events

Syntax:*devpoll_changes*

Default:

These directives specify how many events may be passed to/from kernel, using appropriate method.

The default `devpoll` values are 32, the rest are 512.

multi_accept

Syntax:*multi_accept [on | off]*

Default:*off*

`multi_accept` tries to accept() as many connections as possible after nginx gets notification about a new connection.

rtsig_signo

Syntax:*rtsig_signo*

Default:

nginx uses two signals when the `rtsig` method is used. The directive specified the first signal number. The second is plus 1.

By default `rtsig_signo` is SIGRTMIN+10 (40).

rtsig_overflow_events

rtsig_overflow_test

rtsig_overflow_threshold

Syntax:*rtsig*overflow***

Default:

These directives specifies how to handle rtsig queue overflows. When overflow occurred nginx flushes rtsig queue, then it handles events switching between poll() and rtsig. poll() handles consecutively all unhandled events, while rtsig periodicaly drains queue to prevent a new overflow. When overflow is handled completely, nginx switches to rtsig method again.

The rtsig_overflow_events specifies the number of events to be passed via poll(). The default is 16.

The rtsig_overflow_test specifies after which number of events handled by poll() nginx will drains rtsig queue. The default is 32.

The rtsig_overflow_threshold works in Linux 2.4.x only. Before to drain rtsig queue nginx looks in a kernel how the queue is filled up

The default is 1/10. "rtsig_overflow_threshold 3" means 1/3.

use

Syntax:*use [kqueue | rtsig | epoll | /dev/poll | select | poll | eventport]*

Default:

如果在 `./configure` 的时候指定了不止一种事件模型，那么可以设置其中一个，以便告诉nginx使用哪种事件模型。默认情况下nginx会在 `./configure` 时找出最适合系统的事件模型。

你可以在 这里 查看可用的事件模型以及如何在 `./configure` 时激活

worker_connections

Syntax:*worker_connections number*

Default:

通过worker_connections和worker_proceses可以计算出maxclients：

```
max_clients = worker_processes * worker_connections
```

作为反向代理，max_clients为：

```
max_clients = worker_processes * worker_connections/4
```

Since a browser opens 2 connections by default to a server and nginx uses the fds (file descriptors) from the same pool to connect to the upstream backend

References

鍣烘湰妯″潡

http核心模块

指令

alias

syntax:*alias file-path|directory-path;*

default:*no*

context:*location*

This directive assigns a path to be used for the indicated location. Note that it may look similar to the `root` directive, but the document root doesn't change, just the file system path used for the request.

For example:

```
location  /i/ {
 alias   /spool/w3/images/;
}
```

The request "/i/top.gif" will return the file "/spool/w3/images/top.gif".

It is possible to use variables in the replacement path.

The `alias` directive cannot be used inside a regex-specified `location`. If you need to do this you must use a combination of `rewrite` and `root`.

client_body_in_file_only

syntax:*client_body_in_file_only on|off*

default:*off*

context:*http, server, location*

The directive enables to store a client request body in a file.

Please note that the file at the request completion will not be removed if the directive is enabled.

This directive can be used for debugging and for the `$r->request_body_file` method in the Embedded Perl module.

client_body_in_single_buffer

syntax:*client_body_in_single_buffer*

default:*off*

context:*http, server, location*

The directive(0.7.58+) specifies whether to keep the whole body in a single client request buffer. The directive is recommended when using the variable $request_body to reduce the operations of copying.

client_body_buffer_size

syntax:*client_body_buffer_size the_size*

default:*8k/16k*

context:*http, server, location*

The directive specifies the client request body buffer size.

If the request body is more than the buffer, then the entire request body or some part is written in a temporary file.

The default size is equal to two pages size, depending on platform it is either 8K or 16K.

client_body_temp_path

syntax:*client_body_temp_path dir-path [level1 [level2 [level3]*

default:*client_body_temp*

context:*http, server, location*

The directive assigns the directory for storing the temporary files in it with the body of the request.

In the `dir-path` a hierarchy of subdirectories up to three levels are possible.

For example

```
client_body_temp_path /spool/nginx/client_temp 1 2;
```

The directory structure will be like this:

```
/spool/nginx/client_temp/7/45/00000123457
```

client_body_timeout

syntax:*client_body_timeout time*

default:*60*

context:*http, server, location*

Directive sets the read timeout for the request body from client.

The timeout is set only if a body is not get in one readstep. If after this time the client send nothing, nginx returns error "Request time out" (408).

client_header_buffer_size

syntax:*client_header_buffer_size size*

default:*1k*

context:*http, server*

Directive sets the headerbuffer size for the request header from client.

For the overwhelming majority of requests it is completely sufficient a buffer size of 1K.

However if a big cookie is in the request-header or the request has come from a wap-client the header can not be placed in 1K, therefore, the request-header or a line of request-header is not located completely in this buffer nginx allocate a bigger buffer, the size of the bigger buffer can be set with the instruction large_client_header_buffers.

client_header_timeout

syntax:*client_header_timeout time*

default:*60*

context:*http, server*

Directive assigns timeout with reading of the title of the request of client.

The timeout is set only if a header is not get in one readstep. If after this time the client send nothing, nginx returns error "Request time out" (408).

client_max_body_size

syntax:*client_max_body_size size*

default:*client_max_body_size 1m*

context:*http, server, location*

Directive assigns the maximum accepted body size of client request, indicated by the line `Content-Length` in the header of request.

If size is greater the given one, then the client gets the error "Request Entity Too Large" (413).

It is necessary to keep in mind that the browsers do not know how to correctly show this error.

default_type

syntax:*default_type MIME-type*

default:*default_type text/plain*

context:*http, server, location*

Assigns the default MIME-type to be used for files where the standard MIME map doesn't specify anything.

See also types

Example:

```
location = /proxy.pac {
 default_type application/x-ns-proxy-autoconfig;
}
location = /wpad.dat {

  rewrite . /proxy.pac;
 default_type application/x-ns-proxy-autoconfig;
}
```

directio

syntax:*directio [size|off]*

default:*directio off*

context:*http, server, location*

The directive enables use of flags O_DIRECT (FreeBSD, Linux), F_NOCACHE (Mac OS X) or directio() function (Solaris) for reading files with size greater than specified. This directive disables use of sendfile for this request. This directive may be useful for big files:

```
    directio  4m;
```

error_page

syntax:*error_page code [code...] [= | =answer-code] uri | @named_location*

default:*no*

context:*http, server, location, if in location*

The directive specifies the URI, which will be showed for the errors indicated.

Example:

```
error_page   404          /404.html;

error_page   502 503 504  /50x.html;
error_page   403          http://example.com/forbidden.html;
error_page   404          = @fetch;
```

Furthermore, it is possible to change the code of answer to another, for example:

```
error_page 404 =200 /.empty.gif;
```

If an erroneous answer is processed by the proxied or FastCGI server and this server can return the different answer codes, for example, 200, 302, 401 or 404, then it is possible to hide the code returned:

```
error_page   404   =  /404.php;
```

If you wish to return the error code as-is, OMIT the = from the error_page directive:

```
error_page   404  /404.php;
```

if_modified_since

syntax:*if_modified_since [off|exact|before]*

default:*if_modified_since exact*

context:*http, server, location*

The directive (0.7.24) defines how to compare time of file modification and time in request header "If-Modified-Since":

- off — don't check "If-Modified-Since" request header (0.7.34);
- exact — exact match;
- before — file modification time should be less than time in "If-Modified-Since" request header.

index

syntax:*index file [file...]*

default:*index index.html*

context:*http, server, location*

Directive determines the file(s) which will be used as the index. It's possible to use variables in the name of file. The presence of the files is checked in the order of their enumeration. A file with an absolute path can be put at the end. Example using a variable:

```
index  index.$geo.html  index.0.html  /index.html;
```

If you want to automatically generate an index from a directory listing, useautoindex on.

internal

syntax:*internal*

default:*no*

context:*location*

internal indicates that the matching location can be used only for so called "internal" requests.

For external requests it will return the error "Not found" (404).

Internal requests are the following:

- requests redirected by the instruction **error_page**
- subrequests created by the command **include virtual** of the "ngx_http_ssi_module" module
- requests changed by the instruction **rewrite** of the "ngx_http_rewrite_module" module

An example to prevent clients fetching error pages directly:

```
error_page 404 /404.html;
location  /404.html {
  internal;

}
```

Nginx 0.7.x introduces a new syntax for internal locations: `@location`

Example:

```
location / {
    root /var/www/html;
    error_page 404 @40x;

}

location @40x {
  root /var/www/errors/40x.html;
}
```

keepalive_timeout

syntax:*keepalive_timeout [time]*

default:*keepalive_timeout 75*

context:*http, server, location*

The first parameter assigns the timeout for keep-alive connections with the client. The server will close connections after this time.

The optional second parameter assigns the `time` value in the header `Keep-Alive: timeout=time` of the response. This header can convince some browsers to close the connection, so that the server does not have to. Without this parameter, nginx does not send a `Keep-Alive` header (though this is not what makes a connection "keep-alive").

The parameters can differ from each other.

Notes on how browsers handle the `Keep-Alive` header:

- MSIE and Opera ignore the "Keep-Alive: timeout=" header.
- MSIE keeps the connection alive for about 60-65 seconds, then sends a TCP RST.
- Opera keeps the connection alive for a long time.
- Mozilla keeps the connection alive for N plus about 1-10 seconds.
- Konqueror keeps the connection alive for about N seconds.

keepalive_requests

syntax:*keepalive_requests n*

default:*keepalive_requests 100*

context:*http, server, location*

Number of requests which can be made over a keep-alive connection.

large_client_header_buffers

syntax:*large_client_header_buffers number size*

default:*large_client_header_buffers 4 4k/8k*

context:*http, server*

Directive assigns the maximum number and size of buffers for large headers to read from client request.

The request line can not be bigger than the size of one buffer, if the client send a bigger header nginx returns error "Request URI too large" (414).

The longest header line of request also must be not more than the size of one buffer, otherwise the client get the error "Bad request" (400).

Buffers are separated only as needed.

By default the size of one buffer is equal to the size of page, depending on platform this either 4K or 8K, if at the end of working request connection converts to state keep-alive, then these buffers are freed.

limit_except

syntax:*limit_except methods {...}*

default:*no*

context:*location*

Directive limits HTTP-methods, accessible inside location.

For the limitation can be used the directives of modules ngx_http_access_module and ngx_http_auth_basic_module:

```
limit_except  GET {
  allow  192.168.1.0/32;
  deny   all;

}
```

limit_rate

syntax:*limit_rate speed*

default:*no*

context:*http, server, location, if in location*

Directive assigns the speed of transmission of the answer to client. Speed is assigned in the bytes per second. Limitation works only for one connection, i.e., if client opens 2 connections, then total velocity will be 2 times higher then the limit set.

If it is necessary to limit speed for the part of the clients at the *server* level, based on some kind of condition - then this directive does not apply. Instead you should specify the limit by assigning the value to the $limit_rate variable, as shown below:

```
server {
  if ($slow) {
    set $limit_rate  4k;
  }

}
```

limit_rate_after

syntax:*limit_rate_after time*

default:*limit_rate_after 1m*

context:*http, server, location, if in location*

The directive limits speed only after the first part was sent.

```
limit_rate_after 1m;
limit_rate 100k;
```

listen

syntax:*listen address:port [default [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ssl]]*

default:*listen 80*

context:*server*

The *listen* directive specifies the address and port accepted by the enclosing server {...} block. It is possible to specify only an address, only a port, or a server name as the address.

```
listen 127.0.0.1:8000;

listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 address(0.7.36) are set in square brackets:

```
listen [::]:8000;
listen [fe80::1];
```

When Linux (in contrast to FreeBSD) binds IPv6 [::], it will also bind the corresponding IPv4 address. If other non IPv6 server definitions already used this, the bind will fail. By using explicit addresses instead of [::] this will not be a problem. It is also possible to specify that this listen directive is only to bind the IPv6 address with use of the "default ipv6only=on" option. Note that this only affect this listen directive, the same server {...} block may very well listen to IPv4 as well, specified by other listen directives.

```
listen [2a02:750:5::123]:80;
listen [::]:80 default ipv6only=on;
```

If only address is given, the default port is 80.

If the directive has the *default* parameter, then the enclosing server {...} block will be the default server for the address:port pair. This is useful for name-based virtual hosting where you wish to specify the default server block for hostnames that do not match any [#server_name server_name] directives. If there are no directives with the `default` parameter, then the default server will be the first server block in which the `address:port` pair appears.

The `listen` directive accepts several parameters, specific to the system calls `listen(2)` and `bind(2)`. These parameters must follow the `default` parameter.

backlog=num -- is assigned parameter backlog in call `listen(2)`. By default backlog equals -1.

rcvbuf=size -- assigned to the parameter `SO_RCVBUF` for the listening socket.

sndbuf=size -- assigned to the parameter `SO_SNDBUF` for the listening socket.

accept_filter=filter -- is assigned name accept-filter.

. It works only to FreeBSD, it is possible to use two filters -- `dataready` and `httpready`. On the signal -HUP accept-filter it is possible to change only in the quite last versions FreeBSD: 6.0, 5.4-STABLE and 4.11-STABLE. deferred -- indicates to use that postponed accept(2) on

Linux with

. the aid of option `TCP_DEFER_ACCEPT` . bind -- indicates that it is necessary to make `bind(2)` separately

. for this pair of address:port. The fact is that if are described several directives listen with the identical port, but by different addresses and one of the directives listen listens to on all addresses for this port (*:port), then nginx will make bind(2) only to* :port. It is necessary to consider that in this case for determining the address, on which the connections arrive, is done the system call getsockname(). But if are used parameters backlog, rcvbuf, sndbuf, accept_filter or deferred, then it is always done separately for this pair of address:port bind(2). ssl -- parameter (0.7.14) not related to listen(2) and bind(2) syscalls

. but instead specifies that connections accepted on this port should work in SSL mode. This allows to specify compact configurations for servers working with both HTTP and HTTPS. For example:

```
listen  80;
listen  443 default ssl;
```

Example of the use of the parameters:

```
listen  127.0.0.1 default accept_filter=dataready backlog=1024;
```

location

syntax:*location [=|~|~|^~] /uri/ { ... }*

default:*no*

context:*server*

This directive allows different configurations depending on the URI. It can be configured using both literal strings and regular expressions. To use regular expressions, you must use the a prefix:

1. `~*` for case insensitive matching
2. `~` for case sensitive matching

To determine which *location* directive matches a particular query, the literal strings are checked first. Literal strings match the beginning portion of the query and are case-sensitive - the most specific match will be used (see below on how nginx determines this). Afterwards,

regular expressions are checked in the order defined in the configuration file. The first regular expression to match the query will stop the search. If no regular expression matches are found, the result from the literal string search is used.

There are two ways to modify this behavior. The first is to use the prefix "=", which matches an exact query only. If the query matches, then searching stops and the request is handled immediately. For example, if the request "/" occurs frequently, then using "location = /" will expedite the processing of this request.

The second is to use the prefix `^~`. This prefix is used with a literal string and tells nginx to not check regular expressions if the path provided is a match. For instance, "location `^~` /images/" would halt searching if the query begins with /images/ - all regular expression directives would not be checked.

To summarize, the order in which directives are checked is as follows:

1. Directives with the `=` prefix that match the query exactly. If found, searching stops.
2. All remaining directives with conventional strings. If this match used the `^~` prefix, searching stops.
3. Regular expressions, in the order they are defined in the configuration file.
4. If #3 yielded a match, that result is used. Otherwise, the match from #2 is used.

It is important to know that nginx does the comparison against decoded URIs. For example, if you wish to match "/images/%20/test", then you must use "/images/ /test" to determine the location.

Example:

```
location  = / {

  # matches the query / only.
  [ configuration A ]
}
location  / {
  # matches any query, since all queries begin with /, but regular
  # expressions and any longer conventional blocks will be

  # matched first.
  [ configuration B ]
}
location ^~ /images/ {
  # matches any query beginning with /images/ and halts searching,
  # so regular expressions will not be checked.

  [ configuration C ]
}
location ~* \.(gif|jpg|jpeg)$ {
  # matches any request ending in gif, jpg, or jpeg. However, all
  # requests to the /images/ directory will be handled by

  # Configuration C.
  [ configuration D ]
}
```

Example requests:

- / -> configuration A
- /documents/document.html -> configuration B
- /images/1.gif -> configuration C
- /documents/1.jpg -> configuration D

Note that you could define these 4 configurations in any order and the results would remain the same. While nested locations are allowed by the configuration file parser, their use is discouraged and may produce unexpected results.

How nginx Determines Which Path Matches

Most users will not need to know how nginx internally determines which path to use - know that it will choose the "most specific" match for your URI in a speedy and efficient manner. For those that are curious, however, read on.

All path strings are sorted alphabetically. nginx then proceeds to search down the list looking for matches until the request URI has a "higher" value then the current string in the sorted list. This is determined using the family of `strcmp()` functions - once `strcmp()` returns 1, then searching stops. Once searching stops, the last string which matched is used.

For example, lets say we have the following paths:

```
/
/a
/apple
/banana
```

Now, lets say the server gets the path "/az". nginx would begin search down this list. First, "/" would match, but "/ is less than "/az" so searching continues. "/a" also matches, but "/a" is still less than "/az" so we continue again. "/apple" does not match. The next string, "/banana", is greater than "/az" so searching stops and the last match, "/a", would be used.

Named Locations

Later versions of Nginx (>0.7.x) have Named Locations. These are location blocks that start with an @ symbol, and are treated similar to internal locations except that they preserve the original URI on internal redirects for error_page or try_files directives.

```
location / {

 try_files @joomla index.html;
}
location @joomla {
  rewrite ^(.*)$ /index.php?q=$1 last;

}
```

log_not_found

syntax:*log_not_found [on|off]*

default:*log_not_found on*

context:*http, server, location*

The directive enables or disables messages in error_log about files not found on disk.

log_subrequest

syntax:*log_subrequest [on|off]*

default:*log_subrequest off*

context:*http, server, location*

The directive enables or disables messages in access_log about sub-requests such as rewrite rules and/or SSI requests.

msie_padding

syntax:*msie_padding [on|off]*

default:*msie_padding on*

context:*http, server, location*

This directive enables or disables the the msie_padding feature for MSIE browsers. When this is enabled, nginx will pad the size of the response body to a minimum of 512 bytes, for responses with a status code above or equal to 400.

The padding prevents the activation of "friendly" HTTP error pages in MSIE, so as to not hide the more-informative error pages from the server.

Note that Chromium/Chrome have "friendly" HTTP error pages as well, but this feature will not send them the padding.

msie_refresh

syntax:*msie_refresh [on|off]*

default:*msie_refresh off*

context:*http, server, location*

This directive allows or forbids issuing a `refresh` instead of doing a `redirect` for MSIE.

open_file_cache

syntax:*open_file_cache max = N [inactive = time] | off*

default:*open_file_cache off*

context:*http, server, location*

The directive sets the cache activity on. These information can be stored:

- Open file descriptors, information with their size and modification time;
- Information about the existence of directories;
- Error information when searches for a file - no file, do not have rights to read, etc. See also open_file_cache_errors

Options directive:

- `max` - specifies the maximum number of entries in the cache. When the cache overflows, the longest-used items(LRU) will be removed;
- `inactive` - specifies the time when the cached item is removed, if it has not been downloaded during that time, the default is 60 seconds;
- `off` - prohibits the cache activity.

Example:

```
open_file_cache max=1000 inactive=20s;
open_file_cache_valid    30s;
open_file_cache_min_uses 2;
open_file_cache_errors   on;
```

open_file_cache_errors

syntax:*open_file_cache_errors on | off*

default:*open_file_cache_errors off*

context:*http, server, location*

The directive specifies to cache errors or not when searching a file.

open_file_cache_min_uses

syntax:*open_file_cache_min_uses number*

default:*open_file_cache_min_uses 1*

context:*http, server, location*

The directive defines the minimum use number of a file within the time specified in the directive parameter inactive in open_file_cache. ?If use more than the number, the file descriptor will remain open in the cache.

open_file_cache_valid

syntax:*open_file_cache_valid time*

default:*open_file_cache_valid 60*

context:*http, server, location*

The directive specifies the time when need to check the validity of the information about the item in open_file_cache.

optimize_server_names

syntax:*optimize_server_names [on|off]*

default:*optimize_server_names on*

context:*http, server*

Directive activates or deactivates optimization of host name checks for name-based virtual servers.

In particular, the check influences the name of the host used in redirects. If optimization is on, and all name-based servers listening on one address:port pair have identical configuration, then names are not checked during request execution and redirects use first server name.

If redirect must use host name passed by the client, then the optimization must be turned off.

Note: this directive is deprecated in nginx 0.7.x, use server_name_in_redirect instead.

port_in_redirect

syntax:*port_in_redirect [on|off]*

default:*port_in_redirect on*

context:*http, server, location*

Directive allows or prevents port indication in redirects handled by nginx.

If `port_in_redirect` is on, then Nginx will not add the port in the url when the request is redirected.

recursive_error_pages

syntax:*recursive_error_pages [on|off]*

default:*recursive_error_pages off*

context:*http, server, location*

`recursive_error_pages` enables or disables following a chain of `error_page` directives.

resolver

syntax:*resolver address*

default:*no*

context:*http, server, location*

TODO: Description

resolver_timeout

syntax:*resolver_timeout time*

default:*30*

context:*http, server, location*

Resolver timeout in seconds.

root

syntax:*root path*

default:*root html*

context:*http, server, location, if in location*

root specifies the document root for the requests. For example, with this configuration

```
location  /i/ {
 root  /spool/w3;
}
```

A request for "/i/top.gif" will return the file "/spool/w3/i/top.gif". You can use variables in the argument.

note: Keep in mind that the root will still append the directory to the request so that a request for "/i/top.gif" will not look in "/spool/w3/top.gif" like might happen in an Apache-like alias configuration where the location match itself is dropped. Use the `alias` directive to achieve the Apache-like functionality.

satisfy_any

syntax:*satisfy_any [on|off]*

default:*satisfy_any off*

context:*location*

Directive solves access with at least one successful checking, executed by modules NginxHttpAccessModule orNginxHttpAuthBasicModule:

```
location  / {
 satisfy_any  on;
  allow  192.168.1.0/32;
  deny   all;
  auth_basic            "closed site";
  auth_basic_user_file  conf/htpasswd;

}
```

send_timeout

syntax:*send_timeout the time*

default:*send_timeout 60*

context:*http, server, location*

Directive assigns response timeout to client. Timeout is established not on entire transfer of answer, but only between two operations of reading, if after this time client will take nothing, then nginx is shutting down the connection.

sendfile

syntax:*sendfile [on|off]*

default:*sendfile off*

context:*http, server, location*

Directive activate or deactivate the usage of `sendfile()`.

server

syntax:*server {...}*

default:*no*

context:*http*

Directive assigns configuration for the virtual server.

There is no separation of IP and name-based (the `Host` header of the request) servers.

Instead, the directive `listen` is used to describe all addresses and ports on which incoming connections can occur, and in directive `server_name` indicate all names of the server.

server_name

syntax:*server_name name [...]*

default:*server_name hostname*

context:*server*

This directive performs two actions:

- Compares the `Host` header of the incoming HTTP request against the server { ... } blocks in the Nginx configuration files and selects the first one that matches. This is how **virtual servers** are defined. Server names are processed in the following order:

- full, static names

- names with a wildcard at the start of the name — *.example.com
- names with a wildcard at the end of the name — www.example.*
- names with regular expressions

If there is no match, a [#server server { ... }] block in the configuration file will be used based on the following order:

1. the server block with a matching `listen` directive marked as `default`
2. the first server block with a matching `listen` directive (or implicit `listen 80;`)

3. Sets the server name that will be used in HTTP redirects if server_name_in_redirect is set.

Example:

```
server {

 server_name   example.com  www.example.com;
}
```

The first name becomes the basic name of server. By default the name of the machine (hostname) is used.

It is possible to use "*" for replacing the first or the last part of the name:

```
server {
 server_name   example.com  *.example.com  www.example.*;
}
```

Two of the above given names can be combined into one:

```
server {
 server_name  .example.com;
}
```

It is also possible to use regular expressions in server names, prepending the name with a tilde "~" like so:

```
server {
 server_name   www.example.com   ~^www\d+\.example\.com$;

}
```

The basic name of server is used in an HTTP redirects, if no `Host` header was in client request or that header does not match any assigned server*name. You can also use just "" to force Nginx to use the `Host` header in the HTTP redirect (note that "" cannot be used as the first name, but you can use a dummy name such as "*" instead):

```
server {
 server_name example.com *;

}
server {
 server_name  _ *;
}
```

Note that this has changed in 0.6.x and is now:

```
server {
 server_name _;

}
```

Since nginx 0.7.12, an empty server name is supported, to catch the requests without "Host" header:

```
server {
server_name "";
}
```

server_name_in_redirect

syntax:*server_name_in_redirect on|off*

default:*server_name_in_redirect on*

context:*http, server, location*

If `server_name_in_redirect` is on, then Nginx will use the first value of the server_name directive for redirects. If `server_name_in_redirect` is off, then nginx will use the requested `Host` header.

server_names_hash_max_size

syntax:*server_names_hash_max_size number*

default:*server_names_hash_max_size 512*

context:*http*

The maximum size of the server name hash tables. For more detail see the description of tuning the hash tables in Nginx Optimizations.

server_names_hash_bucket_size

syntax:*server_names_hash_bucket_size number*

default:*server_names_hash_bucket_size 32/64/128*

context:*http*

Directive assigns the size of basket in the hash-tables of the names of servers. This value by default depends on the size of the line of processor cache. For more detail see the description of tuning the hash tables in Nginx Optimizations.

server_tokens

syntax:*server_tokens on|off*

default:*server_tokens on*

context:*http, server, location*

Whether to send the Nginx version number in error pages and `Server` header.

tcp_nodelay

syntax:*tcp_nodelay [on|off]*

default:*tcp_nodelay on*

context:*http, server, location*

This directive allows or forbids the use of the socket option `TCP_NODELAY`. Only included in `keep-alive` connections.

You can read more about the `TCP_NODELAY` socket option here.

tcp_nopush

syntax:*tcp_nopush [on|off]*

default:*tcp_nopush off*

context:*http, server, location*

This directive permits or forbids the use of the socket options `TCP_NOPUSH` on FreeBSD or `TCP_CORK` on Linux. This option is only available when using `sendfile`.

Setting this option causes nginx to attempt to send it's HTTP response headers in one packet on Linux and FreeBSD 4.xReadMoreAboutTcpNopush

try_files

syntax:*try_files file1 [file2 ... filen] fallback*

default:*none*

context:*location*

This directive tells Nginx to test for each file's existence, and use the first found file as the URI. If none of the files are found, then the location `fallback` is called ("fallback" can be any name). `fallback` is a required parameter. It can be a named location or any *guaranteed* URI.

Example:

```
location / {

  try_files index.html index.htm @fallback;
}

location @fallback {
 root  /var/www/error;
 index index.html;

}
```

types

syntax:*types {...}*

context:*http, server, location*

Directive assigns the correspondence of expansion and MIME-types of answers. To one MIME- type can correspond several expansions. By default it is used these correspondences:

```
 types  {

  text/html    html;
  image/gif    gif;
  image/jpeg   jpg;
}
```

The sufficiently complete table of mappings is included and is located in the file `conf/mime.types`.

So that for that determined location's for all answers would reveal MIME- type `application/octet-stream`, it is possible to use the following:

```
location /download/ {
types          { }

 default_type  application/octet-stream;
}
```

变量

The core module supports built-in variables, whose names correspond with the names of variables in Apache.

First of all, there are the variables, which represent the lines of the title of the client request, for example, `$http_user_agent`, `$http_cookie`, and so forth.

Furthermore, there are other variables:

$arg_PARAMETER

This variable contains the value of the `GET` request variable *PARAMETER* if present in the query string

$args

This variable is equal to arguments in the line of request;

$binary_remote_addr

The address of the client in binary form;

$body_bytes_sent

(undocumented)

$content_length

This variable is equal to line `Content-Length` in the header of request;

$content_type

This variable is equal to line `Content-Type` in the header of request;

$cookie_COOKIE

The value of the cookie *COOKIE*;

$document_root

This variable is equal to the value of directive root for the current request;

$document_uri

The same as $uri.

$host

This variable is equal to line `Host` in the header of request or name of the server processing the request if the `Host` header is not available.

$http_HEADER

The value of the HTTP header *HEADER* when converted to lowercase and with 'dashes' converted to 'underscores', e.g. $http_user_agent, $http_referer...;

$is_args

Evaluates to "?" if $args is set, "" otherwise.

$limit_rate

This variable allows limiting the connection rate.

$query_string

The same as $args.

$remote_addr

The address of the client.

$remote_port

The port of the client;

$remote_user

This variable is equal to the name of user, authenticated by the Auth Basic Module;

$request_filename

This variable is equal to path to the file for the current request, formed from directives root or alias and URI request;

$request_body

This variable(0.7.58+) contains the body of the request. The significance of this variable appears in locations with directives proxy_pass or fastcgi_pass.

$request_body_file

Client request body temporary filename;

$request_completion

(undocumented)

$request_method

This variable is equal to the method of request, usually `GET` or `POST` .

$request_uri

This variable is equal to the complete initial URI together with the arguments;

$scheme

The HTTP scheme (i.e. http, https). Evaluated only on demand, for example:

```
rewrite  ^(.+)$  $scheme://example.com$1  redirect;
```

$server_addr

Equal to the server address. As a rule, for obtaining the value of this variable is done one system call. In order to avoid system call, it is necessary to indicate addresses in directives listen and to use parameter `bind` .

$server_name

The name of the server.

$server_port

This variable is equal to the port of the server, to which the request arrived;

$server_protocol

This variable is equal to the protocol of request, usually this `HTTP/1.0` or `HTTP/1.1` .

$uri

This variable is equal to current URI in the request, it can differ from initial, for example by internal redirects, or with the use of index it is file with internal redirects.

References

Original Documentation

HttpIndex模块

这个模块提供一个简单方法来实现在轮询和客户端IP之间的后端服务器负荷平衡。

配置范例：

```
upstream backend  {
  server backend1.example.com weight=5;
  server backend2.example.com:8080;
  server unix:/tmp/backend3;
}

server {
  location / {
    proxy_pass  http://backend;
  }
}
```

配置指导

ip_hash

syntax: ip_hash

default: none

context: upstream

This directive causes requests to be distributed between upstreams based on the IP-address of the client. The key for the hash is the class-C network address of the client. This method guarantees that the client request will always be transferred to the same server. But if this server is considered inoperative, then the request of this client will be transferred to another server. This gives a high probability clients will always connect to the same server.

范例：

```
upstream backend {
  ip_hash;
  server   backend1.example.com;
  server   backend2.example.com;
  server   backend3.example.com  down;
  server   backend4.example.com;
}
```

server

syntax: server name [parameters]

default: none

context: upstream

HttpAccess模块

此模块提供了一个简易的基于主机的访问控制.

ngx_http_access_module 模块使有可能对特定IP客户端进行控制. 规则检查按照第一次匹配的顺序

配置样例

```
location / {
: deny    192.168.1.1;
: allow   192.168.1.0/24;
: allow   10.1.1.0/16;
: deny    all;
}
```

在上面的例子中,仅允许网段 10.1.1.0/16 和 192.168.1.0/24中除 192.168.1.1之外的ip访问.

当执行很多规则时,最好使用 ngx_http_geo_module 模块.

指导

- [#放行 放行]
- [#禁止 禁止]

放行

syntax:*allow [address | CIDR | all]*

default:*no*

context:*http, server, location, limit_except*

以上描述的网络地址有权直接访问

禁止

syntax:*deny [address | CIDR | all]*

default:*no*

context:*http, server, location, limit_except*

以上描述的网络地址拒绝访问

References

Original Documentation

HttpAuthBasic模块

该模块可以使你使用用户名和密码基于 HTTP 基本认证方法来保护你的站点或其部分内容。

实例配置

```
location  /  {
: auth_basic            "Restricted";
: auth_basic_user_file  conf/htpasswd;
}
```

指令

- [#auth_basic auth_basic]
- [#auth_basic_user_file auth_basic_user_file]

auth_basic

语法：*auth_basic [text|off]*

默认值：*auth_basic off*

作用域：*http, server, location, limit_except*

该指令包含用于 HTTP 基本认证 的测试名和密码。分配的参数用于认证领域。值 "off" 可以使其覆盖来自上层指令的继承性。

auth_basic_user_file

语法：*auth_basic_user_file the_file*

默认值：*no*

作用域：*http, server, location, limit_except*

该指令为某认证领域指定 htpasswd 文件名。

文件格式类似于下面的内容：

```
用户名:密码
用户名2:密码2:注释
用户名3:密码3
```

密码必须使用函数 crypt(3) 加密。 你可以使用来自 Apache 的 htpasswd 工具来创建密码文件。

你也可以使用perl 创建密码文件,pw.pl 的内容：

```
#!/usr/bin/perl
use strict;

my $pw=$ARGV[0] ;
print crypt($pw,$pw)."\n";
```

然後執行

```
chmod +x pw.pl
./pw.pl password
papAq5PwY/QQM
```

papAq5PwY/QQM 就是password 的crypt()密码

参考

原始文档

HttpAutoindex模块

此模块用于自动生成目录列表.

ngx_http_autoindex_module只在 ngx_http_index_module模块未找到索引文件时发出请求.

配置实例

```
location  /  {
: autoindex  on;
}
```

指导

- [#autoindex autoindex]
- [#autoindex_exact_size autoindex_exact_size]
- [#autoindex_localtime autoindex_localtime]

autoindex

syntax:*autoindex [on|off]*

default:*autoindex off*

context:*http, server, location*

激活/关闭自动索引

autoindex_exact_size

syntax:*autoindex_exact_size [on|off]*

default:*autoindex_exact_size on*

context:*http, server, location*

设定索引时文件大小的单位(B,KB, MB 或 GB)

autoindex_localtime

syntax:*autoindex_localtime [on|off]*

default:*autoindex_localtime off*

context:*http, server, location*

开启以本地时间来显示文件时间的功能。默认为关（GMT时间）

参考

Original Documentation

Browser模块

摘要

This module creates variables, the values of which depend on the request header "User-agent":

本模块的变量基于请求头(header)中的"User-agent":

- $modern_browser - is equal to the value, assigned by directive modern_browser_value, if browser is identified as an **modern** browser;
- $ancient_browser - 当等于指定给modern_browser_value的浏览器时，这个浏览器被认定为新版的浏览器;
- $ancient_browser - is equal to the value, assigned by directive ancient_browser_value, if browser is identified as an **old** browser;
- $ancient_browser - 当等于指定给ancient_browser_value的浏览器时，这个浏览器被认定为老版的浏览器;
- $msie - is equal **1**, if browser is identified as MSIE with any version;
- $msie - 当浏览器为MSIE的任何版本时，这个值等于1;

If you don't need this module add --without-http_browser_module parameter to the ./configure call, at compile time.

如果不需要这个模块时，在编译的时候加上 --without-http_browser_module.

Example configuration 例如配置

Selection of the index file:

选择索引文件:

```
modern_browser_value "modern.";

modern_browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
modern_browser safari 413;

modern_browser konqueror 3.0;
index index.${modern_browser}html index.html;
```

Redirect for the old browsers:

定义老的浏览器:

```
modern_browser msie 5.0;

modern_browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;

modern_browser unlisted;
ancient_browser Links Lynx Netscape4;

if ($ancient_browser){
  rewrite  ^  /ancient.html;
}
```

指令

ancient_browser

syntax:*ancient_browser line [line...]*

default:*no*

context:*http, server, location*

Directive assigns the substrings, during presence of which in the line "User-agent", browser are considered as **old**.
Special line "netscape4" corresponds to regular expression "^Mozilla/[1-4] ".

ancient_browser_value

syntax:*ancient_browser_value line*

default:*ancient_browser_value 1*

context:*http, server, location*

Directive assigns value for the variables $ancient_browser.

定义 $ancient_browser的变量值.

modern_browser

syntax:*modern_browser browser version|unlisted*

default:*no*

context:*http, server, location*

Directive assigns which version of the browser is to be considered as **modern**.
As browser you can assign the values msie, gecko (Mozilla-based browsers) opera, safari, konqueror.

Of versions it is possible to assign in size X, X.X, X.X.X, or X.X.X.X. maximum values for each of their sizes respectively - 4000, 4000.99, 4000.99.99, and 4000.99.99.99.

Special value "unlisted" indicates to consider modern browser, not described by the modern_browser and ancient_browser directives.
Otherwise the neperechislennyy browser will be considered become obsolete.
If the headers do not contain "User-agent", the browser is considered neperechislennym.

modern_browser_value

syntax:*modern_browser_value line*

default:*modern_browser_value 1*

context:*http, server, location*

Directive assigns value for the variables $modern_browser.

定义$modern_browser的变量值.

References

Original Documentation new since 26.09.2006 version 0.4.3

Charset妯″潡

This module adds the text encoding to the "Content-Type indicated" response-header.

Furthermore, module can reencode data of one encoding into another. It is necessary to note that the reencoding is accomplished only in one direction - from the server to the client, and only one-byte encodings can be reencoded.

Example configuration:

```
charset         windows-1251;
source_charset  koi8-r;
```

鎸囦护

charset

syntax:*charset encoding|off*

default:*charset off*

context:*http, server, location, if in location*

The directive charset adds the line "Content-Type" into response-header with indicated encoding. If this encoding is differed from that indicated in directive source_charset, then reencoding is carried out. The parameter "off" deactivate the insertation of the line "Content-Type" in the response-header.

charset_map

syntax:*charset_map encoding1 encoding2 {...}*

default:*no*

context:*http, server, location*

The directive charset_map describes the table of reencoding from one encoding into another. A table for the inverse reencoding is created using the same data. The codes of symbols are assigned in hexadecimal form. If no recorded symbols are in the range 80-FF they will be substituted with '?'.

Example usage:

```
charset_map  koi8-r  windows-1251 {
  C0  FE ; # small yu
  C1  E0 ; # small a

  C2  E1 ; # small b
  C3  F6 ; # small ts
  # ...
}
```

The complete table of conversion from koi8-r into Windows-1251 is distributed with nginx and is located in file conf/koi-win.

override_charset

syntax:*override_charset on|off*

default:*override_charset off*

context:*http, server, location, if in location*

This directive determines, to carry out reencoding for the response, obtained from the proxied server or from FastCGI-server, if in the response-header a "Content-Type" header already is. If reencoding is permitted, then as the initial encoding the encoding, indicated in the obtained answer, is used.

It is necessary to note that if the response was obtained in the subquery then, independent of directive override_charset, is always carried out reencoding from encoding of the response into encoding of basic demand.

source_charset

syntax:*source_charset encoding*

default:*no*

context:*http, server, location, if in location*

The directive source_charset assigns the initial encoding of response. If this encoding is differed from that indicated in directive charset, then reencoding is carried out.

References

Original Documentation

HttpEmptyGif模块

本模块在内存中常驻了一个 1x1 的透明 GIF 图像，可以被非常快速的调用。

示例：

```
location = /_.gif {
: empty_gif;
}
```

指令

- [#empty_gif empty_gif]

empty_gif >

语法：*empty_gif*

默认值：*n/a*

作用域*location*

参见

原始文档

HttpFcgi模块

这个模块允许Nginx 与FastCGI 进程交互，并通过传递参数来控制FastCGI 进程工作。

配置实例:

```
location / {
  fastcgi_pass   localhost:9000;
  fastcgi_index  index.php;

  fastcgi_param  SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
  fastcgi_param  QUERY_STRING     $query_string;
  fastcgi_param  REQUEST_METHOD   $request_method;
  fastcgi_param  CONTENT_TYPE     $content_type;
  fastcgi_param  CONTENT_LENGTH   $content_length;
}
```

语法：

fastcgi_buffers

```
syntax: fastcgi_buffers the_number is_size;

default: fastcgi_buffers 8 4k/8k;

context: http, server, location

该指令集设置缓冲区的数量和大小，用于缓存从 FastCGI Server 接收到的数据。默认情况下，一个缓冲区的大小相当
```

fastcgi_buffer_size

syntax: fastcgi_buffer_size the_size

default: fastcgi_buffer_size 4k/8k

context: http, server, location

This directive sets the buffersize, into which will be read the first part of the response, obtained from the fastcgi server.

In this part of response the small response-header is located, as a rule.

By default, the buffersize is equal to the size of one buffer in directive fastcgi_buffers; however, it is possible to set it to less.

fastcgi_cache

syntax: fastcgi_cache zone;

default: none

context: http, server, location

设置缓存在共享内存中的名称. 一块区域可以被用于不用的地方.

fastcgi_cache_key

syntax: fastcgi_cache_key line ;

default: none

context: http, server, location

设置缓存的key, 例:

```
fastcgi_cache_key localhost: 9000 $ request_uri;
```

fastcgi_cache_methods

syntax: fastcgi_cache_methods [GET HEAD POST];

default: fastcgi_cache_methods GET HEAD;

context: main,http,location

GET/HEAD is syntax sugar, i.e. you can not disable GET/HEAD even if you set just

```
fastcgi_cache_methods  POST;
```

fastcgi_cache_min_uses

syntax: fastcgi_cache_min_uses n

default: fastcgi_cache_min_uses 1

context: http, server, location

TODO: Description.

fastcgi_cache_path

syntax: fastcgi_cache_path /path/to/cache [levels=m:n keys_zone=name:time inactive=time clean_time=time]

default: none

context: http, server, location

TODO: Description.

fastcgi_cache_use_stale

```
syntax: fastcgi_cache_use_stale [updating|error|timeout|invalid_header|http_500]

default: fastcgi_cache_use_stale off;

context: http, server, location

TODO: Description.
```

fastcgi_cache_valid

```
syntax: fastcgi_cache_valid [http_error_code|time]

default: none

context: http, server, location

TODO: Description.
```

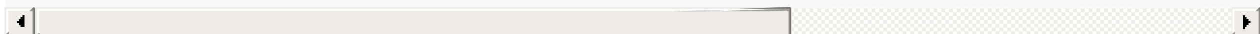
fastcgi_index

```
syntax: fastcgi_index file

default: none

context: http, server, location

The name of the file which will be appended to the URI and stored in the variable $fastcg
```



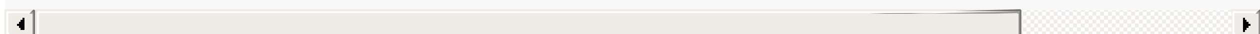
fastcgi_hide_header

```
syntax: fastcgi_hide_header name

context: http, server, location

默认情况下Nginx 不会从FastCGI 进程里给客户端发送"Status" 和"X-Accel-..." 消息头。这个指令可以用来掩

如果需要"Status" 和"X-Accel-..." 消息头，那就需要使用这个指令让FastCGI 强制发送消息头给客户端。
```



fastcgi_ignore_client_abort

```
syntax: fastcgi_ignore_client_abort on|off

default: fastcgi_ignore_client_abort off

context: http, server, location

这个指令用来决定忽略用户取消的请求。
```

fastcgi_intercept_errors

```
syntax: fastcgi_intercept_errors on|off

default: fastcgi_intercept_errors off

context: http, server, location

这个指令用来决定是否要把客户端转向4xx和5xx错误页，或允许Nginx自动指定错误页页。

注意：你需要在此明确错误页,它才是有用的。Igor 曾说：“如果没有定制的处理机制，Nginx不会拦截一个没有缺省页
```

fastcgi_param

```
syntax: fastcgi_param parameter value

default: none

context: http, server, location

该指令指定的参数,将被传递给FastCGI-server。

它可能使用字符串、变量及其它们的组合来作为参数值。如果不在此制定参数，它就会继承外层设置；如果在此设置了参

下面是一个例子,对于PHP来说的最精简的必要参数：

  fastcgi_param  SCRIPT_FILENAME  /home/www/scripts/php$fastcgi_script_name;
  fastcgi_param  QUERY_STRING     $query_string;

参数SCRIPT_FILENAME 是PHP 用来确定执行脚本的名字，而参数QUERY_STRING 是它的一个子参数。

如果要处理POST,那么这三个附加参数是必要的：

  fastcgi_param  REQUEST_METHOD   $request_method;
  fastcgi_param  CONTENT_TYPE     $content_type;
  fastcgi_param  CONTENT_LENGTH   $content_length;

如果PHP 在编译时使用了--enable-force-cgi-redirect选项，设置参数REDIRECT_STATUS 的值为200就是必须

  fastcgi_param  REDIRECT_STATUS  200;
```

Geo模块

摘要

This module creates variables, whose values depend on the IP-address of the client.

Example configuration:

```
geo  $geo  {

  default          0;
  127.0.0.1/32     2;
  192.168.1.0/24   1;
  10.1.0.0/16      1;

}
```

指令

geo

syntax:*geo [$ip_variable] $variable { ... }*

default: none

context:*http*

The directive describes the dependency of the value of a variable on the IP-address of a client. By default, the IP-address used for the lookup is $remote_addr, but since version 0.7.27 it is possible to specify which variable should be used.

```
 geo  $arg_remote_addr $geo {
   ...;
 }
```

Addresses are assigned in the form CIDR. Furthermore, there are four special parameters:

Example of the description:

- delete – deletes the specified network (0.7.23).
- default - the value of variable, if the client address does not correspond to any assigned address. It is possible so to write instead of default 0.0.0.0/0.
- include - text file with addresses and values information. Several files can be inluded like this.
- proxy - specifies the address of proxy server (0.8.7+). NEED MORE DESCRIPTION...
- ranges – specifies that the addresses specified are in the form of ranges (0.7.23). This

directive must be the first.

```
geo  $country  {
  default          no;
  include          conf/geo.conf;
  127.0.0.0/24     us;
  127.0.0.1/32     ru;
  10.1.0.0/16      ru;
  192.168.1.0/24   uk;
}
```

In the file conf/geo.conf:

```
10.2.0.0/16      ru;
192.168.2.0/24   ru;
```

The value will be the the one with maximum agreement. For example, the IP address 127.0.0.1 will get the value "ru", but not "us".

Example with ranges:

```
geo  $country  {
  ranges;
  default                      no;
  127.0.0.0-127.0.0.0          us;
  127.0.0.1-127.0.0.1          ru;
  127.0.0.1-127.0.0.255        us;
  10.1.0.0-10.1.255.255        ru;
  192.168.1.0-192.168.1.255    uk;
}
```

References

- HWLoadbalancerCheckErrors
- Creating geo.conf From MaxMind GeoIP Country Database
- Original Documentation

HttpGzip模块

这个模块支持在线实时压缩输出数据流

使用范例

```
: gzip             on;
: gzip_min_length  1000;
: gzip_proxied     expired no-cache no-store private auth;
: gzip_types       text/plain application/xml;
```

内置变量 $gzip_ratio 可以获取到gzip的压缩比率

指令

- [#gzip gzip]
- [#gzip_buffers gzip_buffers]
- [#gzip_comp_level gzip_comp_level]
- [#gzip_min_length gzip_min_length]
- [#gzip_http_version gzip_http_version]
- [#gzip_proxied gzip_proxied]
- [#gzip_types gzip_types]

gzip

语法:*gzip on|off*

默认值:*gzip off*

作用域:*http, server, location, if (x) location*

开启或者关闭gzip模块

gzip_buffers

语法:*gzip_buffers number size*

默认值:*gzip_buffers 4 4k/8k*

作用域:*http, server, location*

设置系统获取几个单位的缓存用于存储gzip的压缩结果数据流。 例如 4 4k 代表以4k为单位，按照原始数据大小以4k为单位的4倍申请内存。 4 8k 代表以8k为单位，按照原始数据大小以8k为单位的4倍申请内存。

如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储gzip压缩结果。

gzip_comp_level

语法:*gzip_comp_level 1..9*

默认值:*gzip_comp_level 1*

作用域:*http, server, location*

gzip压缩比，1 压缩比最小处理速度最快，9 压缩比最大但处理最慢（传输快但比较消耗cpu）。

gzip_min_length

语法:*gzip_min_length length*

默认值:*gzip_min_length 0*

作用域:*http, server, location*

设置允许压缩的页面最小字节数，页面字节数从header头中的Content-Length中进行获取。

默认值是0，不管页面多大都压缩。

建议设置成大于1k的字节数，小于1k可能会越压越大。 即: gzip_min_length 1024

gzip_http_version

语法:*gzip_http_version 1.0|1.1*

默认值:*gzip_http_version 1.1*

作用域:*http, server, location*

识别http的协议版本。由于早期的一些浏览器或者http客户端，可能不支持gzip自解压，用户就会看到乱码，所以做一些判断还是有必要的。 注：21世纪都来了，现在除了类似于百度的蜘蛛之类的东西不支持自解压，99.99%的浏览器基本上都支持gzip解压了，所以可以不用设这个值,保持系统默认即可。

gzip_proxied

语法:*gzip_proxied [off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any] ...*

默认值:*gzip_proxied off*

作用域:*http, server, location*

Nginx作为反向代理的时候启用，开启或者关闭后端服务器返回的结果，匹配的前提是后端服务器必须要返回包含"Via"的 header头。

- off - 关闭所有的代理结果数据的压缩
- expired - 启用压缩，如果header头中包含 "Expires" 头信息
- no-cache - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息
- no-store - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息
- private - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息
- no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
- no_etag - 启用压缩 ,如果header头中不包含 "ETag" 头信息
- auth - 启用压缩 , 如果header头中包含 "Authorization" 头信息
- any - 无条件启用压缩

gzip_types

语法:*gzip_types mime-type [mime-type ...]*

默认值:*gzip_types text/html*

作用域:*http, server, location*

匹配MIME类型进行压缩，（无论是否指定）"text/html"类型总是会被压缩的。

注意：如果作为http server来使用，主配置文件中要包含文件类型配置文件

```
http
{
        include       conf/mime.types;
        ......
}
```

如果你希望压缩常规的文件类型，可以写成这个样子

```
http
{
: include       conf/mime.types;

: gzip on;
: gzip_min_length  1000;
: gzip_buffers     4 8k;
: gzip_http_version 1.1;
: gzip_types       text/plain application/x-javascript text/css text/html application/xml

: ......
}
```

References

原始文档

HttpHeaders模块

本模板可以设置HTTP报文的头标。

示例

```
: expires     24h;
: expires     0;
: expires     -1;
: expires     epoch;
: add_header  Cache-Control  private;
```

指令

- [#add_header add_header]
- [#expires expires]

增加头标

语法：*add_header name value*

默认值：*none*

作用域：*http, server, location*

当HTTP应答状态码为 200、204、301、302 或 304 的时候，增加指定的HTTP头标。

其中头标的值可以使用变量。

expires

语法：*expires [time|epoch|max|off]*

默认值：*expires off*

作用域：*http, server, location*

使用本指令可以控制HTTP应答中的“Expires”和“Cache-Control”的头标，（起到控制页面缓存的作用）。

可以在time值中使用正数或负数。“Expires”头标的值将通过当前系统时间加上您设定的 time 值来获得。

`epoch` 指定“Expires”的值为 1 January, 1970, 00:00:01 GMT。

`max` 指定“Expires”的值为 31 December 2037 23:59:59 GMT，“Cache-Control”的值为10年。

`-1` 指定“Expires”的值为 服务器当前时间 -1s,即永远过期

“Cache-Control”头标的值由您指定的时间来决定：

- 负数：`Cache-Control: no-cache`
- 正数或零：`Cache-Control: max-age = #`, # 为您指定时间的秒数。

"off" 表示不修改“Expires”和“Cache-Control”的值

References

原始文档

HttpIndex模块

语法:*index file [file...]*

默认值:*index index.html*

作用域:*http, server, location*

该指令用来指定用来做默认文档的文件名，可以在文件名处使用变量。 如果您指定了多个文件，那么将按照您指定的顺序逐个查找。 可以在列表末尾加上一个绝对路径名的文件。

示例:

```
index  index.$geo.html  index.0.html  /index.html;
```

参见

原始文档

HttpLimit zone

本模块可以针对条件，进行会话的并发连接数控制。（例如：限制每个IP的并发连接数。）

配置示例

```
http {
: limit_zone   one  $binary_remote_addr  10m;

: ...

: server {

: ...

: location /download/ {
: limit_conn   one  1;
: }
```

指令

- [#limit_zone limit_zone]
- [#limit_conn limit_conn]

limit_zone

语法：*limit_zone zone_name $variable the_size*

默认值：*no*

作用域：*http*

本指令定义了一个数据区，里面记录会话状态信息。
$variable 定义判断会话的变量；the_size 定义记录区的总容量。

例子：

```
limit_zone   one  $binary_remote_addr  10m;
```

定义一个叫“one”的记录区，总容量为 10M，以变量 $binary_remote_addr 作为会话的判断基准（即一个地址一个会话）。

您可以注意到了，在这里使用的是 $binary_remote_addr 而不是 $remote_addr。

$remote_addr 的长度为 7 至 15 bytes，会话信息的长度为 32 或 64 bytes。 而 $binary_remote_addr 的长度为 4 bytes，会话信息的长度为 32 bytes。

当区的大小为 1M 的时候，大约可以记录 32000 个会话信息（一个会话占用 32 bytes）。

limit_conn

语法：*limit_conn zone_name the_size*

默认值：*no*

作用域：*http, server, location*

指定一个会话最大的并发连接数。 当超过指定的最发并发连接数时，服务器将返回 "Service unavailable" (503)。

例子：

```
limit_zone   one  $binary_remote_addr  10m;

: server {
: location /download/ {
: limit_conn   one  1;
: }
```

定义一个叫“one”的记录区，总容量为 10M，以变量 $binary_remote_addr 作为会话的判断基准（即一个地址一个会话）。 限制 /download/ 目录下，一个会话只能进行一个连接。 简单点，就是限制 /download/ 目录下，一个IP只能发起一个连接，多过一个，一律503。

References

原始文档

HttpLimitReqest妯″潡

This module allows you to limit the number of requests for a given session, or as a special case, with one address.

Restriction done using leaky bucket.

Example Configuration

```
http {
    limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;


    ...

    server {

        ...

        location /search/ {

            limit_req   zone=one  burst=5;
        }
```

鎸囦护

Syntax:*limit_req_zone **$session_variable** zone=**name_of_zone**:**size** rate=**rate***

Default:*none*

Context:*http*

The directive describes the area, which stores the state of the sessions. The values of the sessions is determined by the given variable. Example of usage:

```
limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;
```

In this case, the session state is allocated 10MB as a zone called "one", and the average speed of queries for this zone is limited to 1 request per second.

The sessions are tracked per-user in this case, but note that instead of the variable $remote_addr, we've used the variable $binary_remote_addr, reducing the size of the state to 64 bytes. A 1 MB zone can hold approximately 16000 states of this size.

The speed is set in requests per second or requests per minute. The rate must be an integer, so if you need to specify less than one request per second, say, one request every two seconds, you would specify it as "30r/m".

Syntax:*limit_req zone=**zone** burst=**burst** [nodelay]*

Default:*none*

Context:*http, server, location*

The directive specifies the zone (**zone**) and the maximum possible bursts of requests (burst). If the rate exceeds the demands outlined in the zone, the request is delayed, so that queries are processed at a given speed. Excess requests are delayed until their number does not exceed a specified number of bursts. In this case the request is completed the code "Service unavailable" (503). By default, the burst is zero.

For example, the directive

```
limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;


    server {
        location /search/ {
            limit_req   zone=one  burst=5;
        }
```

allows a user no more than 1 request per second on average, with bursts of no more than 5 queries. If the excess requests within the limit burst delay are not necessary, you should use the nodelay:

```
            limit_req   zone=one  burst=5  nodelay;
```

HttpLog模块

ngx_http_log_module 实例

```
log_format  gzip  '$remote_addr - $remote_user [$time_local]  '
: '"$request" $status $bytes_sent '
: '"$http_referer" "$http_user_agent" "$gzip_ratio"';
access_log  /spool/logs/nginx-access.log  gzip  buffer=32k;
```

指令

- access_log
- log_format

access_log

语法：*access_log path [format [buffer=size | off *]* 默认值：access_log log/access.log combined*

作用域：*http, server, location*

指令 access_log 指派路径、格式和缓存大小。参数 "off" 将清除当前级别的所有 access_log 指令。如果未指定格式，则使用预置的 "combined" 格式。缓存不能大于能写入磁盘的文件的最大大小。在 FreeBSD 3.0-6.0， 缓存大小无此限制。

log_format

语法：*log_format name format [format ...]*

默认值：*log_format combined "..."*

作用域：*http*server*

Directive log_format describes the format of a log entry. Besides general variables in the format it is possible to use variables which exist only at the moment of record into the log:

- $body_bytes_sent, the number of bytes, transmitted to client minus the response headers, variable is compatible with parameter %B of module Apache's mod_log_config (this was called $apache_bytes_sent, before version 0.3.10)
- $bytes_sent, the number of bytes, transmitted to client
- $connection, the number of connection
- $msec, the time with an accuracy to microseconds at the moment of the log entry
- $pipe, "p" if request was pipelining
- $request_length, the length of the body of the request

- $request_time, the time of working on request in seconds
- $status, status of answer
- $time_local, local time into common log format.

The headers, transmitted to client, begin from the prefix "sent*http*", for example, $sent_http_content_range.

In the configuration there is always a predetermined format "combined":

```
log_format  combined  '$remote_addr - $remote_user [$time_local]  '
: '"$request" $status $apache_bytes_sent '
: '"$http_referer" "$http_user_agent"';
```

参考

原始文档

map

This module allows you to classify, or map a set of values into a different set of values and store the result in a variable.

Example:

```
map  $http_host  $name  {
  hostnames;


  default          0;

  example.com      1;
  *.example.com    1;
  test.com         2;
  *.test.com       2;
  .site.com        3;

}
```

One use for this would be to use a mapping in place of writing lots of server/location directives or redirects:

```
map $uri $new {
  default        http://www.domain.com/home/;

  /aa            http://aa.domain.com/;
  /bb            http://bb.domain.com/;
  /john          http://my.domain.com/users/john/;

}

server {
  server_name   www.domain.com;
  rewrite  ^    $new   redirect;
}
```

指令

map

syntax:*map $var1 $var2 { ... }*

default:*none*

context:*http*

map defines the mapping table which will be used to set a variable. There are three special parameters:

- default — defines the value to be used where no match is found.

- hostnames — it allows for an easier matching of values like host names, names with a starting dot may match exact host names and host names ending with the value, for example:

```
*.example.com  1;
```

Instead of two entries

```
example.com    1;
*.example.com  1;
```

we can use only one

```
.example.com   1;
```

- include — include values from a file. Multiple includes may be used.

map_hash_max_size

syntax:*map_hash_max_size number*

default:*map_hash_max_size 2048*

context:*http*

The directive sets the maximum size of a hash table to hold the variable map. For more details see the descriptions of hash settings Optimization section .

map_hash_bucket_size

syntax:*map_hash_bucket_size n*

default:*map_hash_bucket_size 32/64/128*

context:*http*

The directive sets the maximum size in a hash table to map variables. The default value depends on the size of the cache line processor. More see in the descriptions of hash settings in the Optimization section .

References

Original Documentation

Memcached

你可以利用本模块来进行简单的缓存以提高系统效率。本模块计划在未来进行扩展。

配置示例

```
server {
: location / {
: set  $memcached_key  $uri;
: memcached_pass   name:11211;
: default_type     text/html;
: error_page       404 = /fallback;
: }

: location = /fallback {
: proxy_pass       backend;
: }
}
```

指令

- [#memcached_pass memcached_pass]
- [#memcached_connect_timeout memcached_connect_timeout]
- [#memcached_send_timeout memcached_send_timeout]
- [#memcached_read_timeout memcached_read_timeout]
- [#memcached_buffer_size memcached_buffer_size]
- [#memcached_next_upstream memcached_next_upstream]

变量

- $memcached_key

memcached_pass

语法：*memcached_pass [name:port]*

默认值：*none*

作用域：*http, server, location*

The backend should set the data in memcached. The memcached key is "/uri?args".

Since 0.5.9 the memcached key is now in `$memcached_key`.

memcached_connect_timeout

语法：*memcached_connect_timeout [time]*

默认值：*60000*

作用域：*http, server, location*

The timeout for connecting to memcached, in milliseconds.

memcached_read_timeout

语法：*memcached_read_timeout [time]*

默认值：*60000*

作用域：*http, server, location*

The timeout for reading from memcached, in milliseconds.

memcached_send_timeout

语法：*memcached_send_timeout [time]*

默认值：*60000*

作用域：*http, server, location*

The timeout for sending to memcached, in milliseconds.

memcached_buffer_size

语法：*memcached_buffer_size [size]*

默认值： see *getpagesize(2)*

作用域：*http, server, location*

The recv/send buffer size, in bytes.

memcached_next_upstream

语法：*memcached_next_upstream [error | timeout | invalid_response | not_found | off]*

默认值：*error timeout*

作用域：*http, server, location*

Which failure conditions should cause the request to be forwarded to another upstream server? Applies only when the value in memcached_pass is an upstream with two or more servers.

HttpProxy模块

This module makes it possible to transfer requests to another server. 此模块专伺将请求导向其它服务.

It is an HTTP/1.0 proxy without the ability for keep-alive requests yet. 这是种 HTTP/1.0 版本的无请求保持代理.

(As a result, backend connections are created and destroyed on every request.) Nginx talks HTTP/1.1 to the browser and HTTP/1.0 to the backend server. As such it handles keep-alive to the browser.

(因为每个请求都是在后台连接中创建和销毁的) Nginx 和浏览器使用 HTTP/1.1 进行对话，而在后台服务中使用 HTTP/1.0;

示例

```
location / {
: proxy_pass        http://localhost:8000;
: proxy_set_header  X-Real-IP  $remote_addr;
}
```

注意一点,当使用HTTP PROXY 模块时(或者甚至是使用FastCGI时),用户的整个请求会在nginx中缓冲直至传送给后端被代理的服务器.因此,上传进度的测算就会运作得不正确,如果它们通过测算后端服务器收到的数据来工作的话

定向器. Directives

- [#proxy_buffer_size proxy_buffer_size]
- [#proxy_buffering proxy_buffering]
- [#proxy_buffers proxy_buffers]
- [#proxy_busy_buffers_size proxy_busy_buffers_size]
- [#proxy_connect_timeout proxy_connect_timeout]
- [#proxy_headers_hash_bucket_size proxy_headers_hash_bucket_size]
- [#proxy_headers_hash_max_size proxy_headers_hash_max_size]
- [#proxy_hide_header proxy_hide_header]
- [#proxy_ignore_client_abort proxy_ignore_client_abort]
- [#proxy_intercept_errors proxy_intercept_errors]
- [#proxy_max_temp_file_size proxy_max_temp_file_size]
- [#proxy_method proxy_method]
- [#proxy_next_upstream proxy_next_upstream]
- [#proxy_pass proxy_pass]
- [#proxy_pass_header proxy_pass_header]

- [#proxy_pass_request_body proxy_pass_request_body]
- [#proxy_pass_request_headers proxy_pass_request_headers]
- [#proxy_redirect proxy_redirect]
- [#proxy_read_timeout proxy_read_timeout]
- [#proxy_redirect_errors proxy_redirect_errors]
- [#proxy_send_lowat proxy_send_lowat]
- [#proxy_send_timeout proxy_send_timeout]
- [#proxy_set_body proxy_set_body]
- [#proxy_set_header proxy_set_header]
- [#proxy_store proxy_store]
- [#proxy_store_access proxy_store_access]
- [#proxy_temp_file_write_size proxy_temp_file_write_size]
- [#proxy_temp_path proxy_temp_path]
- [#proxy_upstream_fail_timeout proxy_upstream_fail_timeout]
- [#proxy_upstream_max_fails proxy_upstream_max_fails]

参数.Variables

- [#var_proxy_host var_proxy_host]
- [#var_proxy_port var_proxy_port]
- [#var_proxy_add_x_forwarded_for var_proxy_add_x_forwarded_for]
- [#var_proxy_remote_addr var_proxy_remote_addr]

proxy_buffer_size

语法:*proxy_buffer_size the_size*

默认值:*proxy_buffer_size 4k/8k*

上下文:*http, server, location*

该指令设置缓冲区大小,从被代理的后端服务器取得的响应内容,会先读取放置到这里.

小的响应header通常位于这部分响应内容里边.

默认来说,该缓冲区大小等于指令 `proxy_buffers` 所设置的;但是,你可以把它设置得更小.

proxy_buffering

语法:*proxy_buffering on|off*

默认值:*proxy_buffering on*

上下文:*http, server, location*

该指令开启从后端被代理服务器的响应内容缓冲.

如果缓冲区开启,nginx假定被代理的后端服务器会以最快速度响应,并把内容保存在由指令 `proxy_buffer_size` 和 `proxy_buffers` 指定的缓冲区里边.

如果响应内容无法放在内存里边,那么部分内容会被写到磁盘上.

如果缓冲区被关闭了,那么响应内容会按照获取内容的多少立刻同步传送到客户端

nginx不尝试计算被代理服务器整个响应内容的大小,nginx能从服务器接受的最大数据,是由指令 `proxy_buffer_size` 指定的.

proxy_buffers

语法:*proxy_buffers the_number is_size;*

默认值:*proxy_buffers 8 4k/8k;*

上下文:*http, server, location*

该指令设置缓冲区的大小和数量,从被代理的后端服务器取得的响应内容,会放置到这里. 默认情况下,一个缓冲区的大小等于页面大小,可能是4K也可能是8K,这取决于平台

proxy_busy_buffers_size

语法:*proxy_busy_buffers_size size;*

默认值:*proxy_busy_buffers_size proxy_buffer_size *2;*

上下文:*http, server, location, if*

TODO: Description.

proxy_connect_timeout

语法:*proxy_connect_timeout timeout_in_seconds*

上下文:*http, server, location*

This directive assigns a timeout for the connection to the proxyserver. This is not the time until the server returns the pages, this is the [#proxy_read_timeout proxy_read_timeout] statement. If your proxyserver is up, but hanging (e.g. it does not have enough threads to process your request so it puts you in the pool of connections to deal with later), then this statement will not help as the connection to the server has been made. It is necessary to keep in mind that this time out cannot be more than 75 seconds.

proxy_headers_hash_bucket_size

语法:*proxy_headers_hash_bucket_size size;*

默认值:*proxy_headers_hash_bucket_size 64;*

上下文:*http, server, location, if*

This directive sets the bucket size of the hashtable.
TODO: Better description

proxy_headers_hash_max_size

语法:*proxy_headers_hash_max_size size;*

默认值:*proxy_headers_hash_max_size 512;*

上下文:*http, server, location, if*

This directive sets the maximum size of the hashtable.
TODO: Better description

proxy_hide_header

语法:*proxy_hide_header the_header*

上下文:*http, server, location*

nginx does not transfer the "Date", "Server", "X-Pad" and "X-Accel-..." header lines from the proxied server response. The `proxy_hide_header` directive allows to hide some additional header lines. But if on the contrary the header lines must be passed, then the `proxy_pass_header` should be used. For example if you want to hide the MS-OfficeWebserver and the AspNet-Version:

```
location / {
: proxy_hide_header X-AspNet-Version;
: proxy_hide_header MicrosoftOfficeWebServer;
}
```

This directive can also be very helpful when using X-Accel-Redirect. For example, you may have one set of backend servers which return the headers for a file download, which includes X-Accel-Redirect to the actual file, as well as the correct Content-Type. However, the Redirect URL points to a filleserver which hosts the actual file you wish to serve, and that server sends its own Content-Type header, which might be incorrect, and overrides the header sent by the original backend servers. You can avoid this by adding the proxy_hide_header directive to the fileserver. Example:

```
location / {
: proxy_pass http://backend_servers;
}

location /files/ {
: proxy_pass http://fileserver;
: proxy_hide_header Content-Type;
}
```

proxy_ignore_client_abort

语法:*proxy_ignore_client_abort [on|off]*

默认值:*proxy_ignore_client_abort off*

上下文:*http, server, location*

Available since: 0.3.36
如果客户端断开请求,也保持后端的下载

proxy_intercept_errors

语法:*proxy_intercept_errors [on|off]*

默认值:*proxy_intercept_errors off*

上下文:*http, server, location*

This directive decides if nginx will intercept responses with HTTP status codes of 400 and higher.

By default all responses will be sent as-is from the proxied server.

If you set this to `on` then nginx will intercept status codes that are explicitly handled by an `error_page` directive. Responses with status codes that do not match an `error_page` directive will be sent as-is from the proxied server.

proxy_max_temp_file_size

语法:*proxy_max_temp_file_size size;*

默认值:*proxy_max_temp_file_size 1G;*

上下文:*http, server, location, if*

Available since: 0.1.8
TODO: Description.

proxy_method

语法:*proxy_method [method]*

默认值:*None*

上下文:*http, server, location*

Used to allow the proxying of additional HTTP methods.

Note: at this time, Nginx only appears to allow a single instance of this directive and it only accepts a single argument (method) so it's not clear how useful this might be for proxying to things like Subversion.

Example:

```
: proxy_method PROPFIND;
```

proxy_next_upstream

语法:*proxy_next_upstream [error|timeout|invalid_header|http_500|http_503|http_404|off]*

默认值:*proxy_next_upstream error timeout*

上下文:*http, server, location*

Directive determines, in what cases the request will be transmitted to the next server:

- error — an error has occurred while connecting to the server, sending a request to it, or reading its response;
- timeout — occurred timeout during the connection with the server, transfer the requst or while reading response from the server;
- invalid_header — server returned a empty or incorrect answer;
- http_500 — server returned answer with code 500
- http_503 — server returned answer with code 503
- http_404 — server returned answer with code 404
- off — it forbids the request transfer to the next server

Transferring the request to the next server is only possible when nothing has been transferred to the client -- that is, if an error or timeout arises in the middle of the transfer of the request, then it is not possible to retry the current request on a different server.

proxy_pass

语法:*proxy_pass URL*

默认值:*no*

上下文:*location, if in location*

This directive sets the port or socket, on which listens to the proxied server, and the URI, to which will be reflected location.

Port can be indicated in the form of the name of hostname or address and port, for example,

```
proxy_pass http://localhost:8000/uri /;
```

and socket -- in the form of unix of socket:

```
proxy_pass http://unix:/tmp/backend.socket:/uri /;
```

Path is indicated after the word `unix` and is concluded between two colons.

With the transfer of request to server part URI, which corresponds to location, is substituted to URI, indicated in directive proxy_pass.

But there are two exceptions to this rule, when it is not possible to determine that replaced location:

- if the location is assigned by regular expression;
- if inside proxied location with the help of directive rewrite changes URI and with this configuration will be precisely processed request (break):

```
location  /name/ {
: rewrite      /name/([^/] +)  /users?name=$1  break;
: proxy_pass   http://127.0.0.1;
}
```

For these cases of URI it is transferred without the mapping.

Furthermore, it is possible to indicate so that URI demand it would be transferred in the same form, as it sent client, but not v in the processed form.

During the working:

- two or by more slashes are converted into one slash: "//" -- "/";
- references to the current directory are removed: "/./" -- "/";
- references to the previous catalog are removed: "/dir /../" -- "/".

If on server it is necessary to transmit URI in the unprocessed form, then for this in directive proxy_pass it is necessary to indicate URL server without URI:

```
location  /some/path/ {
: proxy_pass   http://127.0.0.1;
}
```

proxy_pass_header

语法:*proxy_pass_header the_name*

上下文:*http, server, location*

This directive allows transferring header-lines forbidden for response.

For example:

```
location / {
: proxy_pass_header Server;
: proxy_pass_header X-MyHeader;
}
```

proxy_pass_request_body

语法:*proxy_pass_request_body [on | off] ;*

默认值:*proxy_pass_request_body on;*

上下文:*http, server, location*

Available since: 0.1.29
TODO: Description.

proxy_pass_request_headers

语法:*proxy_pass_request_headers [on | off] ;*

默认值:*proxy_pass_request_headers on;*

上下文:*http, server, location*

Available since: 0.1.29
TODO: Description.

proxy_redirect

语法:*proxy_redirect [default|off|redirect replacement]*

默认值:*proxy_redirect default*

上下文:*http, server, location*

This directive sets the text, which must be changed in response-header "Location" and "Refresh" in the response of the proxied server.

Let us suppose the proxied server returned line `Location: http://localhost:8000/two/some/uri/`.

The directive

```
proxy_redirect   http://localhost:8000/two/   http://frontend/one/;
```

will rewrite this line in the form `Location: http://frontend/one/some/uri/`.

In the replaceable line it is possible not to indicate the name of the server:

```
proxy_redirect http://localhost:8000/two/ /;
```

then the basic name of server and port is set, if it is different from 80.

The change by default, given by the parameter "default", uses the parameters of directives location and proxy_pass.

Therefore two following configurations are equivalent:

```
location /one/ {
: proxy_pass       http://upstream:port/two/;
: proxy_redirect   default;
}

location /one/ {
: proxy_pass       http://upstream:port/two/;
: proxy_redirect   http://upstream:port/two/   /one/;
}
```

In the replace line, it is possible to use some variables:

```
proxy_redirect   http://localhost:8000/    http://$host:$server_port/;
```

This directive repeated some times:

```
: proxy_redirect   default;
: proxy_redirect   http://localhost:8000/    /;
: proxy_redirect   http://www.example.com/   /;
```

The parameter `off` forbids all `proxy_redirect` directives at this level:

```
: proxy_redirect   off;
: proxy_redirect   default;
: proxy_redirect   http://localhost:8000/    /;
: proxy_redirect   http://www.example.com/   /;
```

With the help of this directive it is possible to add the name of host for relative redirect, issued by the proxied server:

```
proxy_redirect   /   /;
```

proxy_read_timeout

语法:*proxy_read_timeout the_time*

默认值:*proxy_read_timeout 60*

上下文:*http, server, location*

This directive sets the read timeout for the response of the proxied server. It determines how long NGINX will wait to get the response to a request. The timeout is established not for entire response, but only between two operations of reading.

In contrast to [#proxy_connect_timeout proxy_connect_timeout] , this timeout will catch a server that puts you in it's connection pool but does not respond to you with anything beyond that. Be careful though not to set this too low, as your proxyserver might take a longer time to respond to requests on purpose (e.g. when serving you a report page that takes some time to compute). You are able though to have a different setting per location, which enables you to have a higher proxy_read_timeout for the report page's location.

If the proxied server nothing will communicate after this time, then nginx is shut connection.

proxy_redirect_errors

Deprecated. Use `proxy_intercept_errors` .

proxy_send_lowat

语法:*proxy_send_lowat [on | off]*

默认值:*proxy_send_lowat off;*

上下文:*http, server, location, if*

This directive set SO_SNDLOWAT.
This directive is only available on FreeBSD

proxy_send_timeout

语法:*proxy_send_timeout time_in_seconds*

默认值:*proxy_send_timeout 60*

上下文:*http, server, location*

This directive assigns timeout with the transfer of request to the proxy server. Time out is established not on entire transfer of request, but only between two operations of record. If after this time the proxy server will not take new data, then nginx is shut the connection

proxy_send_lowat

语法:*proxy_set_body [on | off]*

默认值:*proxy_set_body off;*

上下文:*http, server, location, if*

Available since: 0.3.10
TODO: Description.

proxy_set_header

语法:*proxy_set_header header value*

默认值:*Host and Connection*

上下文:*http, server, location*

This directive allows to redefine and to add some request header lines which will be transferred to the proxied server.

As the value it is possible to use a text, variables and their combination.

This directive is inherited from the previous level when at this level are not described their directives `proxy_set_header` .

By default only two lines will be redefined:

```
proxy_set_header Host $proxy_host;
proxy_set_header Connection Close;
```

The unchanged request-header "Host" can be transmitted like this:

```
proxy_set_header Host $http_host;
```

However, if this line is absent from the client request, then nothing will be transferred.

In this case it is better to use variable $host, it's value is equal to the name of server in the request-header "Host" or to the basic name of server, if there is no line:

```
proxy_set_header Host $host;
```

Furthermore, it is possible to transmit the name of server together with the port of the proxied server:

```
proxy_set_header Host $host:$proxy_port;
```

proxy_store

语法:*proxy_store [on | off | path]*

默认值:*proxy_store off*

上下文:*http, server, location*

This directive sets the path in which upstream files are stored. The parameter "on" preserves files in accordance with path specified in directives *alias* or *root*. The parameter "off" forbids storing. Furthermore, the name of the path can be clearly assigned with the aid of the line with the variables:

```
proxy_store   /data/www$original_uri;
```

The time of modification for the file will be set to the date of "Last-Modified" header in the response. To be able to safe files in this directory it is necessary that the path is under the directory with temporary files, given by directive proxy_temp_path for the data location.

This directive can be used for creating the local copies for dynamic output of the backend which is not very often changed, for example:

```
location /images/ {
: root                 /data/www;
: error_page           404 = /fetch$uri;
}

location /fetch {
: internal;

: proxy_pass           http://backend;
: proxy_store          on;
: proxy_store_access   user:rw  group:rw  all:r;
: proxy_temp_path      /data/temp;

: alias                /data/www;
}
```

To be clear proxy_store is not a cache, it's rather mirror on demand.

proxy_store_access

语法:*proxy_store_access users:permissions [users:permission ...]*

默认值:*proxy_store_access user:rw*

上下文:*http, server, location*

This directive assigns the permissions for the created files and directories, for example:

```
proxy_store_access  user:rw  group:rw  all:r;
```

If any rights for groups or all are assigned, then it is not necessary to assign rights for user:

```
proxy_store_access  group:rw  all:r;
```

proxy_temp_file_write_size

语法:*proxy_temp_file_write_size size;*

默认值:*proxy_temp_file_write_size proxy_buffer_size *2;*

上下文:*http, server, location, if*

TODO: Description.

proxy_temp_path

语法:*proxy_temp_path dir-path [level1 [level2 [level3] ;*

默认值:*$NGX_PREFIX/proxy_temp controlled by --http-proxy-temp-path at ./configure stage*

上下文:*http, server, location*

This directive works like client_body_temp_path to specify a location to buffer large proxied requests to the filesystem.

proxy_upstream_fail_timeout

Changed in 0.5.0 to deprecated.
Please use the *fail_timeout* parameter of server Directive from the upstream module.

proxy_upstream_max_fails

Changed in 0.5.0 to deprecated.
Please use the *max_fails* parameter of server Directive from the upstream module.

Variables

In module ngx_http_proxy_module there are some built-in variables, which can be used for the creation of headers with the help of the `proxy_set_header` directive:

$proxy_host:: the name of proxied host and port;

$proxy_port:: the port of proxied host;

$proxy_add_x_forwarded_for:: equivalent to client request-header "X-Forwarded-For" and to variable added to it through the comma

$remote_addr:: But if there is no line "X-Forwarded-For" in the client request, then variable `$proxy_add_x_forwarded_for` is equal to variable `$remote_addr` .

References

Original Documentation

Questions

- Is it possible to have the proxy use a client SSL certificate to communicate with the server being proxied? I am in the situation where I want to secure the proxy <-> application server connection. Another solution could be to use SSL only with a custom header checked at each connection in the application server, but doing it at the protocol level would be nicer.

- NginxHttpProxyModule#preview 原文

HttpRewrite模块

This module makes it possible to change URI using regular expressions, and to redirect and select configuration depending on variables.

该模块允许使用正则表达式改变URI，并且根据变量来转向以及选择配置。

If the directives of this module are given at the server level, then they are carried out before the location of the request is determined. If in that selected location there are further rewrite directives, then they also are carried out. If the URI changed as a result of the execution of directives inside location, then location is again determined for the new URI.

如果在server级别设置该选项，那么他们将在location之前生效。如果在location还有更进一步的重写规则，location部分的规则依然会被执行。如果这个URI重写是因为location部分的规则造成的，那么location部分会再次被执行作为新的URI。

This cycle can be repeated up to 10 times, after which Nginx returns a 500 error.

这个循环会执行10次，然后Nginx会返回一个500错误。

Directives

- [#break break]
- [#if if]
- [#return return]
- [#rewrite rewrite]
- [#set set]
- [#uninitialized_variable_warn uninitialized_variable_warn]

break

语法:*break*

默认值:*none*

作用域:*server, location, if*

Completes the current set of rules. 作用是完成当前的规则列

示例:

```
if ($slow) {
: limit_rate  10k;
: break;
}
```

if

语法:*if (condition) { ... }*

默认:*none*

作用域:*server, location*

Checks the truth of a condition. If the condition evaluates to true, then the code indicated in the curly braces is carried out and the request is processed in accordance with the configuration within the following block. Configuration inside directive if is inherited from the previous level.

They can be assigned as the condition:

- the name of variable; false values are: empty string "", or any string starting with "0";
- the comparison of variable with the line with using the `=` and `!=` operators;
- pattern matching with regular expressions using the symbols `~*` and `~`:
- `~` is case-sensitive match;
- '~'符合，但是大小写敏感
- `~*` specifies a case-insensitive match (firefox matches FireFox)
- ~*大小写不敏感的符合（firefox符合FireFox）
- `!~ and !~*` mean the opposite, "doesn't match"
- !~和!~*代表相反，“不符合”
- checking for the existence of a file using the `-f and !-f` operators;使用-f以及!-f检测一个文件是否存在
- checking existence of a directory using the `-d and !-d` operators;使用-d以及!-d检测一个目录是否存在
- checking existence of a file, directory or symbolic link using the `-e and !-e` operators; 使用-e以及!-e检测是否存在一个文件，一个目录或者一个符号链接。

- checking whether a file is executable using the `-x and !-x` operators.使用-x以及!-x检测一个文件是否可执行

Parts of the regular expressions can be in parentheses, whose value can then later be accessed in the `$1 to >$9` variables.

Examples of use:

```
if ($http_user_agent ~ MSIE) {
: rewrite  ^(.*)$  /msie/$1  break;
}
if ($http_cookie ~* "id=([^;] +)(?:;|$)" ) {
: set  $id  $1;
}
if ($request_method = POST ) {
: return 405;
}
if (!-f $request_filename) {
: break;
: proxy_pass  http://127.0.0.1;
}
if ($slow) {
: limit_rate  10k;
}
if ($invalid_referer) {
: return   403;
}
```

The value of the built-in variable `$invalid_referer` is given by the directive valid_referers.

return

语法:*return code*

默认值:*none*

作用域:*server, location, if*

This directive concludes execution of the rules and returns the status code indicated to client. It is possible to use the following values: 204, 400, 402-406, 408, 410, 411, 413, 416 and 500-504. Furthermore, nonstandard code 444 closes the connection without sending any headers.

这个指令根据规则的执行情况，返回一个状态值给客户端。可使用值包括：204，400，402-406，408，410，411，413，416以及500-504。也可以发送非标准的444代码-未发送任何头信息下结束连接。

rewrite

语法:*rewrite regex replacement flag*

默认:*none*

作用域:*server, location, if*

This directive changes URI in accordance with the regular expression and the replacement string. Directives are carried out in order of appearance in the configuration file.

这个指令根据表达式来更改URI，或者修改字符串。指令根据配置文件中的顺序来执行。

Be aware that the rewrite regex only matches the relative path instead of the absolute URL. If you want to match the hostname, you should use an if condition, like so:

注意重写表达式只对相对路径有效。如果你想配对主机名，你应该使用if语句，如下：

```
if ($host ~* www\.(.*)) {
: set $host_without_www $1;
: rewrite ^(.*)$ http://$host_without_www$1 permanent; # $1 contains '/foo', not 'www.myd
}
```

Flags make it possible to end the execution of rewrite directives.

If the replacement string begins with `http://` then the client will be redirected, and any further rewrite directives are terminated.

Flags can be any of the following:

- last - completes processing of rewrite directives, after which searches for corresponding URI and location
- break - completes processing of rewrite directives
- redirect - returns temporary redirect with code 302; it is used if the substituting line begins with `http://`
- permanent - returns permanent redirect with code 301

Note that if an redirect is relative (has no host part), then when redirecting Nginx uses the "Host" header if the header match name of server_name directive or the first name of `server_name` directive, if the header does not match or is absent. If no `server_name` is set, then the local hostname is used. If you want Nginx to always use the "Host" header, you can use a wildcard "*" `server_name` (but see the restrictions on doing so).Example:

```
rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  last;
rewrite  ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   last;
return   403;
```

But if we place these directives in location /download/, then it is necessary to replace flag "last" by "break", otherwise nginx will hit the 10 cycle limit and return error 500:

```
location /download/ {
: rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
: rewrite  ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   break;
: return   403;
}
```

If in the line of replacement arguments are indicated, then the rest of the request arguments are appended to them. To avoid having them appended, place a question mark as the last character:

```
: rewrite  ^/users/(.*)$  /show?user=$1?  last;
```

注: 对花括号({ 和 })来说, 他们既能用在重定向的正则表达式里，也是用在配置文件里分割代码块, 为了避免冲突, 正则表达式里带花括号的话，应该用双引号（或者单引号）包围。比如，要将类似以下的url

```
/photos/123456
```

重定向到：

```
/path/to/photos/12/1234/123456.png
```

可以用以下方法 (注意双引号):

```
rewrite  "/photos/([0-9] {2})([0-9] {2})([0-9] {2})" /path/to/photos/$1/$1$2/$1$2$3.png;
```

set

syntax:*set variable value*

default:*none*

context:*server, location, if*

Directive establishes value for the variable indicated. As the value it is possible to use a text, variables and their combination.

uninitialized_variable_warn

syntax:*uninitialized_variable_warn on|off*

default:*uninitialized_variable_warn on*

context:*http, server, location, if*

Enables or disables logging of warnings about noninitialized variables.

Internally, the rewrite directives are compiled at the time the configuration file is loaded into internal codes, usable during the request by the interpreter.

This interpreter is a simple stack virtual machine. For example, the directive:

```
location /download/ {
: if ($forbidden) {
: return   403;
: }
: if ($slow) {
: limit_rate  10k;
: }
: rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
}
```

will be compiled into this sequence:

```
: variable $forbidden
: checking to zero
: recovery 403
: completion of entire code
: variable $slow
: checking to zero
: checkings of regular expression
: copying "/"
: copying $1
: copying "/mp3/"
: copying $2
: copying "..mpe"

: completion of regular expression
: completion of entire sequence
```

Note that there is no code for directive limit_rate, since it does not refer to module ngx_http_rewrite_module. The "if" block exists in the same part of the configuration as the "location" directive.

If $slow is true, then what's inside the "if" block is evaluated, and in this configuration limit_rate it is equal to 10k.

Directive:

```
: rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
```

It is possible to reduce the sequence, if in the regular expression we include the first slash inside the parentheses:

```
: rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
```

then the sequence will appear like this:

```
: checking regular expression
: copying $1
: copying "/mp3/"
: copying $2
: copying "..mpe"
: completion of regular expression
: completion of entire code
```

References

Original Documentation

HttpSSI模块

此模块处理服务器端包含文件(ssi)的处理. 列表中的命令当前并未完全支持.

配置示例

```
location / {
: ssi  on;
}
```

Directives

- [#ssi ssi]
- [#ssi_silent_errors ssi_silent_errors]
- [#ssi_types ssi_types]
- [#ssi_value_length ssi_value_length]

ssi

语法:*ssi [on | off]*

默认值:*ssi off*

作用域:*http, server, location* 在location作用域中将启用SSI文件处理.

ssi_silent_errors

语法:*ssi_silent_errors [on|off]*

默认值:*ssi_silent_errors off*

作用域:*http, server, location*

在处理SSI文件出错时不输出错误提示:"[an error occurred while processing the directive] "

ssi_types

语法:*ssi_types mime-type [mime-type ...]*

默认值:*ssi_types text/html*

作用域:*http, server, location*

Enables SSI processing for MIME-types in addition to "text/html" types.

ssi_value_length

语法:*ssi_value_length length*

默认值:*ssi_value_length 256*

作用域:*http, server, location*

定义SSI允许使用的参数长度

SSI 命令

格式示例如下:

```
: <!--#  command parameter1=value parameter2=value... -->
```

支持的SSI 命令如下:

- `block` — command describes the block, which can be used as a silencer in command `include` . Inside the block there can be commands `SSI` .
- *name* — the name of the block. For example:

: : the silencer : :

- `config` — assigns some parameters with working SSI.
- *errmsg* — the line, which is derived with the error during the SSI processing. By default, this string is used: "[an error occurred while processing the directive] "
- *timefmt* — the time formatting string, as used in strftime(3). By default, this string is used:

: "%A, %d-%b-%Y %H:%M:%S %Z" : To include time in seconds use the format "%s" as well.

- `echo` - print a variable
- *var* — the name of the variable
- *default* - if the variable is empty, display this string. Defaults to "none". Example:

: : </code> is the same as `: no :`

- `if / elif / else / endif` — conditionally include text or other directives. Usage:

.........Only one level of nesting is possible.

- *expr* — the expression to evaluate. It can be a variable: ~~~

```
<!--# if expr="$name" --></code> A string comparison: <code><!--# if expr="$name = text"

- <code>include — include a document from another source.
- *file* — include a file, e.g.
```

```
: <!--# include file="footer.html" --> :

- *virtual* — include a request, e.g.

<!--# include virtual="/remote/body.php?argument=value" -->Multiple requests will be issu

- *stub* — The name of the block to use as a default if the request is empty or returns a

<!--# block name="one" --> <!--# endblock --><!--# include virtual="/remote/body.php?argu

- *wait* — when set to yes, the rest of the SSI will not be evaluated until the current r
```

~~~

- `set` - assign a variable.
- var — the variable.
- value — its value. If it contains variable names, these will be evaluated.

内置变量

ngx_http_ssi_module 支持两种内置变量:

- `$date_local` - 当前的本地时区时间.配置选项"timefmt"控制格式.
- `$date_gmt` - 当前的GMT时间. 配置选项"timefmt"控制格式.

参考

Original Documentation
~~~

HttpUserId

The module ngx_http_userid_module gives out cookies for identification of clients. For logging it is possible to use variables `$uid_got` and `$uid_set`. Remark: keep in mind variables $uid_got and $uid_set are not accessible in SSI, because SSI filter module working before userid filter.

This module is compatible with mod_uid for Apache.

Example

```
userid          on;
userid_name     uid;

userid_domain   example.com;
userid_path     /;
userid_expires  365d;
userid_p3p      'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

鐫囦护

userid

syntax:*userid [on|v1|log||off]*

default:*userid off*

context:*http, server, location*

Enables or disables issuing cookies and logging requested cookies:

- on - enables version 2 cookies and logs them;
- v1 - enables version 1 cookies and logs them;
- log - do not send cookies, but write down incoming cookies to log;
- off - do not send cookies, and don't write them to logs;

userid_domain

syntax:*userid_domain [name | none]*

default:*userid_domain none*

context:*http, server, location*

Assigns the domain for cookie. The parameter "none" doesn't issue domain for cookie.

userid_expires

syntax:*userid_expires [time | max]*

default:*none*

context:*http, server, location*

Sets the expiration time for the cookie.

The parameter set & send-out browser expiration time for cookie. Value "max" assigns the time on 31 December, 2037, 23:55:55 gmt. This is the maximum time that older browsers understand.

userid_name

syntax:*userid_name name*

default:*userid_name uid*

context:*http, server, location*

Assigns name to cookie.

userid_p3p

syntax:*userid_p3p line*

default:*none*

context:*http, server, location*

Directive assigns value for the header P3P, which will sent together with cookie.

userid_path

syntax:*userid_path path*

default:*userid_path /*

context:*http, server, location*

Sets the cookie path.

userid_service

syntax:*userid_service number*

default:*userid_service address*

context:*http, server, location*

Directive assigns the IP address of the server which gave out cookie. If not set, version 1 cookies set to zero, and for version 2 cookies the IP address of server.

References

Original Documentation

鍏朵粬妯″潡

Addition妯″潡

This module adds responses of other locations before and after the current location's response.

It is implemented as an output filter, the contents of the main request and subrequests to other locations are not buffered completely and are still sent to the client in a streaming fashion. Because the length of the final response body is unkown while sending the HTTP headers, the HTTP chunked encoding is always used here.

Installation

By default the module is not built, it is necessary to enable its build with

```
./configure --with-http_addition_module
```

at compilation time.

Example:

```
location / {
  add_before_body   /before_action;
  add_after_body    /after_action;

}
```

Limitations

Note that as of 0.8.17 no contents will be added if the current location is served as a subrequest itself. Consider the following example:

```
location /foo {
  add_before_body /bar;
}


location /bar {
 add_before_body  /baz;
}
```

Then accessing */foo* won't get */baz* inserted before the contents of the subrequest */bar*.

Also note that at this time, only strings can be used in before/after body locations, not variables. So

```
location / {
  set $before_action /before_action;
  add_before_body $before_action;
}
```

will not work as expected (although the configuration file will still load properly).

Directives

add_before_body

syntax:*add_before_body uri*

default:*no*

context:*http, server, location*

Directive adds content of uri before the response body, issued as a result of the work of the assigned subrequest.

add_after_body

syntax:*add_after_body uri*

default:*no*

context:*http, server, location*

Directive adds content of uri after the response body, issued as a result of the work of the assigned subrequest.

addition_types

syntax:*addition_types mime-type [mime-type ...]*

default:*text/html*

context:*http, server, location*

Directive (since 0.7.9) allows you to add text only to locations of the specified MIME-types (defaults to *"text/html"*).

(Before 0.8.17, this directive was mispelled as "addtion_types" in the source. This bug has been fixed in 0.8.17.)

References

Original Documentation

EmbeddedPerl

This module makes it possible to execute Perl directly within Nginx and call Perl via SSI.

Building Module at Compile-time

Unless built at compile-time, the module is not available. The default is to not build this module. If you want to enable this module, is necessary to specify **--with-http_perl_module** when running *configure*.

Your system must have Perl 5.6.1 or above.

Known Problems

This module is experimental; therefore anything is possible and bugs are likely.

1. If a Perl module performs protracted operation, (for example DNS lookups, database queries, etc), then the worker process that is running the Perl script is completely tied up for the duration of script. Therefore embedded Perl scripts should be extremely careful to limit themselves to short, predictable operations.
2. It's possible for Nginx to leak memory if you reload the configuration file (via 'kill -HUP ').

Example:

```
http {

  perl_modules  perl/lib;
  perl_require  hello.pm;

  perl_set  $msie6  '
  sub {
    my $r = shift;
    my $ua = $r->header_in("User-Agent");
    return "" if $ua =~ /Opera/;
    return "1" if $ua =~ / MSIE [6-9] \.\d+/;
    return "";
  }
 ';


  server {
    location / {
      perl  hello::handler;
    }
  }

}
```

perl/lib/hello.pm:

```
package hello;
use nginx;

sub handler {
  my $r = shift;
  $r->send_http_header("text/html");
  return OK if $r->header_only;


  $r->print("hello!\n<br/>");
  $r->rflush;

  if (-f $r->filename or -d _) {

    $r->print($r->uri, " exists!\n");
  }

  return OK;

}

1;
__END__
```

指令

perl

syntax:*perl module::function | 'sub {...}'*

default:*no*

context:*location*

Directive establishes a function for the data location.

perl_modules

syntax:*perl_modules path*

default:*no*

context:*http*

Directive assigns additional path for Perl modules. Since version 0.6.7 this path is relative to directory of nginx configuration file nginx.conf, but not to nginx prefix directory.

perl_require

syntax:*perl_require module*

default:*no*

context:*http*

There can be multiple perl_require directives.

perl_set

syntax:*perl_set module::function | 'sub {...}'*

default:*no*

context:*http*

Directive establishes the function of variable ???

Calling Perl from SSI

Instruction format is as follows:

```
<!- # perl sub="module::function" arg="parameter1" arg="parameter2"... >
```

Methods of request object $r:

- $r->args - method returns the arguments of request.
- $r->filename - method returns the name of file, which corresponds to URI request.
- $r->has_request_body(function) - method returns 0, if there is no request body. But if the request body exists, then the passed function is established and 1 returned. At the end of the body processing, nginx will call the established processor. Example usage:

```
package hello;

use nginx;

sub handler {

  my $r = shift;

  if ($r->request_method ne "POST") {

    return DECLINED;
  }

  if ($r->has_request_body(hello::post)) {
    return OK;
  }


  return 400;
}

sub post {
  my $r = shift;
  $r->send_http_header;
  $r->print("request_body: \"", $r->request_body, "\"<br/>");
  $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>\n");
  return OK;

}

1;

__END__
```

- $r->header_in(header) - retrieve an HTTP request header.
- $r->header_only - true if we only need to return a response header.
- $r->header_out(header, value) - set a response header.
- $r->internal_redirect(uri) - method makes internal redirect to the indicated uri. Redirect occurs only after the completion of Perl script.
- $r->print(args, ...) - sends data to client.
- $r->request_body - method returns the request body of client when the body is not recorded into the temporary file.

So that the body of the demand of client is guaranteed to remain in memory, it is necessary to limit its size with the aidof client_max_body_size and to assign sufficient size for the buffer using client_body_buffer_size.

- $r->request_body_file — method returns the name of the file, in which is stored the body of the demand of client. The file must be removed on the completion of work. So that the body of the request would always be written to the file, it is necessary to indicate client_body_in_file_only on.
- $r->request_method — method returns the HTTP method of the request.
- $r->remote_addr - method returns IP address of client.
- $r->rflush - method immediately transmits data to client.
- $r->sendfile(file [, displacement [, length]) - transfers to client contents of the file

indicated. The optional parameters indicate initial offset and length of transmitted data. Strictly the transmission of data occurs only after the completion of the perl script.

- $r->send_http_header(type) - adds header to response. The optional parameter "type" establishes the value of line "Content-Type" in the title of answer.
- $r->unescape(text) - decodes the text, assigned in the form %XX.
- $r->uri - returns request URI.

References

Original Documentation

flv

This module provides the ability to seek within FLV (Flash) files using time-based offsets.

本模块提供FLV文件加载基于时间位移.

Module ngx_http_flv_module offers special handling of files it handles:

模块ngx_http_flv_module提供特殊处理的文件,它处理:

- adds FLV header to the requested file;
- 添加 FLV 头请求的文件;
- transfers file, beginning from the displacement, specified in the request argument start=XXX.
- 传输文件，从开始的位移，在请求参数中指定 start=XXX.

This module is not compiled by default and must be specified using the --with-http_flv_module argument to configure when compiling Nginx.

本模块必需在编译nginx时加上--with-http_flv_module.

例如：

```
location ~ \.flv$ {

  flv;
}
```

HttpGzipStatic

Before serving a file from disk to a gzip-enabled client, this module will look for a precompressed file in the same location that ends in ".gz". The purpose is to avoid compressing the same file each time it is requested.

在开始压缩创建硬盘上的文件之前，本模块将查找同目录下同名的.gz压缩文件，以避免同一文件再次压缩。

ngx_http_gzip_static_module was introduced in nginx 0.6.24. You must enable support at compile time:

nginx 0.6.24开始加入ngx_http_gzip_static_module . 编译时加上:

```
./configure --with-http_gzip_static_module
```

例如：

```
gzip_static on;

gzip_http_version   1.1;
gzip_proxied        expired no-cache no-store private auth;
gzip_disable        "MSIE [1-6]\.";

gzip_vary           on;
```

指令

gzip_static

syntax:*gzip_static on|off*

default:*gzip_static off*

context:*http, server, location*

Enables the module. You should ensure that the timestamps of the compressed and uncompressed files match.

启动模块。您应该确保压缩和解压文件的时间戳匹配。

gzip_http_version

See NginxHttpGzipModule

gzip_proxied

See NginxHttpGzipModule

gzip_disable

See NginxHttpGzipModule

gzip_vary

See NginxHttpGzipModule

RandomIndex

Pick a random directory index from a directory.

从目录中选择一个随机目录索引。 .

例如:

```
location  /  {

  random_index  on;
}
```

HttpGeoIP

This module creates ngx_http_geoip_module variables based on the IP-address of the client matched against the MaxMind GeoIP binary files. This module appeared in nginx version 0.8.6.

本模块ngx_http_geoip_module的变量基于IP地址匹配MaxMind GeoIP 二进制文件. 这个模块开始出现在nginx0.8.6。

Precondition 首先

This module needs the geo databases and the library to read the database.

模块必需有geo数据库和读取数据库类

```
#下载免费的geo_city数据库
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
#下载免费的geo_coundty数据库
wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz
#在debian中安装libgeoip:
sudo apt-get install libgeoip-dev
#其它系统，你可以下载并编译一个源文件
wget http://geolite.maxmind.com/download/geoip/api/c/GeoIP.tar.gz
```

在centos可以用yum安装:

```
yum install geoip
```

编译

```
./configure --with-http_geoip_module
```

例如

```
http {
    geoip_country  GeoIP.dat;
    geoip_city     GeoLiteCity.dat;
    ...
```

指令

geoip_country

syntax:*geoip_country path/to/db.dat;*

default: none

context:*http*

The directive indicates the path to the .dat file used for determining the visitor's country from the IP-address of the client. When set the module makes available the following variables:

dat文件用于判断访问者IP中的国家。当前模块 下可用的变量：

- $geoip_country_code; -国家名的前两个字母, 如, "RU", "US".
- $geoip_country_code3; - 国家名的前三个字母, 如, "RUS", "USA".
- $geoip_country_name; -国家名称, 如, "Russian Federation", "United States".

If you only need the country's name, you can just set the geoip_country database(1.1M), while the geoip_city database is much bigger(43M) and all the databses will be cached in memory.

如果你只需要国家，你只需设置geoip_country数据库(1.1M), 但城市的ip数据库就比较大(43M)并且将加载到内存当缓存。

geoip_city

syntax:*geoip_city path/to/db.dat;*

default: none

context:*http*

The directive indicates the path to the .dat file used for determining countries, regions and cities from IP-address of the client. When set the module makes available the following variables:

dat文件用于判断访问者IP中的国家、省，城市。当前模块 下可用的变量:

- $geoip_city_country_code; -国家名的前两个字母, 如, "RU", "US".
- $geoip_city_country_code3; - 国家名的前三个字母, 如, "RUS", "USA".
- $geoip_city_country_name; -国家名称, 如, "Russian Federation", "United States".
- $geoip_region; - 省，州或区名 (province, region, state, province, federal land, and the like), 如, "Moscow City", "DC".
- $geoip_city; - 城市名称, 如, "Moscow", "Washington".
- $geoip_postal_code; - 邮政编号.

References

Original Documentation

HttpRealIp

This module lets to change the client's IP address to value from request header (e. g. `X-Real-IP` or `X-Forwarded-For`).

It is useful if nginx works behind some proxy of L7 load balanver, and request come from local IP, but proxy add request header with client's IP.

This module isn't built by default, enable it with the configure option

```
--with-http_realip_module
```

User Note: "You will build a list of trusted proxies (see below) and the first IP in the header which is not trusted will be used as the client IP." Source: README of the Apache module mod_extract . Quite informative, about why and how this security feature is helpful.

Example:

```
set_real_ip_from   192.168.1.0/24;
set_real_ip_from   192.168.2.1;
real_ip_header     X-Real-IP;
```

鎸囦护

set_real_ip_from

syntax:*set_real_ip_from [the address|CIDR]*

default: none

context:*http, server, location*

This directive describes the trusted addresses, which transfer accurate address for the replacement.

real_ip_header

syntax:*real_ip_header [X-Real-IP|X-Forwarded-For]*

default:*real_ip_header X-Real-IP*

context:*http, server, location*

This directive sets the name of the header used for transferring the replacement IP address.

References

Original Documentation

HttpSSL

This module enables HTTPS support.

It supports checking client certificates with two limitations:

- it is not possible to assign the list of the abolished certificates (revocation lists)
- if you have a chain certificate file (sometimes called an intermediate certificate) you don't specify it separately like you do in Apache. Instead you need to add the information from the chain cert to the end of your main certificate file. This can be done by typing "cat chain.crt >> mysite.com.crt" on the command line. Once that is done you won't use the chain cert file for anything else, you just point Nginx to the main certificate file.

By default the module is not built, it is necessary to specify that it be built with with the *--with-http_ssl_module* parameter to ./configure. Building this module requires the OpenSSL libraries and include-files, often are the necessary files in separate packages.

The following is an example configuration, to reduce the CPU load it is recommended to run one worker process only and to enable keep-alive connections:

```
worker_processes 1;
http {

  server {
    listen               443;
    ssl                  on;
    ssl_certificate      /usr/local/nginx/conf/cert.pem;
    ssl_certificate_key  /usr/local/nginx/conf/cert.key;
    keepalive_timeout    70;
  }

}
```

When using chain certificates, just append the extra certificates into your .crt file (cert.pem in the example). Your own certificate needs to be on top of the file, otherwise key get a mismatch with the key.

Generate Certificates

To generate dummy certficates you can do this steps:

```
$ cd /usr/local/nginx/conf
$ openssl genrsa -des3 -out server.key 1024
$ openssl req -new -key server.key -out server.csr
$ cp server.key server.key.org
$ openssl rsa -in server.key.org -out server.key
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Configure the new certificate into nginx.conf:

```
server {

    server_name YOUR_DOMAINNAME_HERE;
    listen 443;
    ssl on;
    ssl_certificate /usr/local/nginx/conf/server.crt;
    ssl_certificate_key /usr/local/nginx/conf/server.key;

}
```

Restart Nginx.

Now all ready to access using:

```
https://YOUR_DOMAINNAME_HERE
```

鎸囦护

ssl

syntax:*ssl [on|off]*

default:*ssl off*

context:*main, server*

Enables HTTPS for a server.

ssl_certificate

syntax:*ssl_certificate file*

default:*ssl_certificate cert.pem*

context:*main, server*

Indicates file with the certificate in PEM format for this virtual server. The same file can contain other certificates, and also secret key in PEM format. Since version 0.6.7 the file path is relative to directory of nginx configuration file nginx.conf, but not to nginx prefix directory.

ssl_certificate_key

syntax:*ssl_certificate_key file*

default:*ssl_certificate_key cert.pem*

context:*main, server*

Indicates file with the secret key in PEM format for this virtual server. Since version 0.6.7 the filename path is relative to directory of nginx configuration file nginx.conf, but not to nginx prefix directory.

ssl_client_certificate

syntax:*ssl_client_certificate file*

default:*none*

context:*main, server*

Indicates file with certificates CA in PEM format, utilized for checking the client certificates.

ssl_dhparam

syntax:*ssl_dhparam file*

default:*none*

context:*main, server*

Indicates file with Diffie-Hellman parameters in PEM format, utilized for negotiating TLS session keys.

ssl_ciphers

syntax:*ssl_ciphers file*

default:*ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP*

context:*main, server*

Directive describes the permitted ciphers. Ciphers are assigned in the formats supported by OpenSSL, for example:

```
ssl_ciphersALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

Complete list can be looked with the following command:

```
openssl ciphers
```

ssl_crl

syntax:*ssl_crl file*

default:*none*

context:*http, server*

This directive (0.8.7+) specifies a file with the revoked certificates (CRL) in the PEM, which is used to check for client certificates.

ssl_prefer_server_ciphers

syntax:*ssl_prefer_server_ciphers [on|off]*

default:*ssl_prefer_server_ciphers off*

context:*main, server*

Requires protocols SSLv3 and TLSv1 server ciphers be preferred over the client's ciphers.

ssl_protocols

syntax:*ssl_protocols [SSLv2] [SSLv3] [TLSv1]*

default:*ssl_protocols SSLv2 SSLv3 TLSv1*

context:*main, server*

Directive enables the protocols indicated.

ssl_verify_client

syntax:*ssl_verify_client on|off|ask*

default:*ssl_verify_client off*

context:*main, server*

Directive enables verifying client certificates. Parameter 'ask' checks a client certificate if it was offered.

ssl_verify_depth

syntax:*ssl_verify_depth number*

default:*ssl_verify_depth 1*

context:*main, server*

Sets depth checking in the chain of client certificates.

ssl_session_cache

syntax:*ssl_session_cache off|none|builtin:size and/or shared:name:size*

default:*ssl_session_cache off*

context:*main, server*

The directive sets the types and sizes of caches to store the SSL sessions.
The cache types are:

- off -- Hard off: nginx says explicitly to a client that sessions can not reused.
- none -- Soft off: nginx says to a client that session can be resued, but nginx actually never reuses them. This is workaround for some mail clients as ssl_session_cache may be used in mail proxy as well as in HTTP server.
- builtin -- the OpenSSL builtin cache, is used inside one worker process only. The cache size is assigned in the number of the sessions. Note: there appears to be a memory fragmentation issue using this method, please take that into consideration when using this. See "References" below.
- shared -- the cache is shared between all worker processes. The size of cache is assigned in the bytes, 1 MB cache can contain about 4000 sessions. Each shared cache must have arbitrary name. Cache with the same name can be used in several virtual servers.

It is possible to use both types of cache simultaneously, for example:

```
ssl_session_cache  builtin:1000  shared:SSL:10m;
```

However, the only shared cache usage without that builtin should be more effective.

ssl_session_timeout

syntax:*ssl_session_timeout time*

default:*ssl_session_timeout 5m*

context:*main, server*

Assigns the time during which the client can repeatedly use the parameters of the session, which is stored in the cache.

This module supports several nonstandard error codes which can be used for debugging with the aid of directive error_page:

- 495 - error checking client certificate
- 496 - client did not grant the required certificate
- 497 - normal request was sent to HTTPS

Debugging is done after the request is completely dismantled and are accessible via variables such as $request_uri, $uri, $arg and others. Built-in variables Module ngx_http_ssl_module supports several built-in variables:

- $ssl_cipher returns the line of those utilized it is cipher for established SSL-connection
- $ssl_client_serial returns the series number of client certificate for established SSL-connection
- $ssl_client_s_dn returns line subject DN of client certificate for established SSL-connection
- $ssl_client_i_dn returns line issuer DN of client certificate for established SSL-connection
- $ssl_protocol returns the protocol of established SSL-connection

ssl_engine

syntax:*ssl_engine*

This allows specifying the OpenSSL engine to use, like Padlock for example. It requires a more recent version of OpenSSL.

References

Original DocumentationSSL Memory Fragmentation and new default status for ssl_session_cache

StubStatus模块

这个模块能够获取Nginx自上次启动以来的工作状态

此模块非核心模块，需要在编译的时候手动添加编译参数 `--with-http_stub_status_module`

配置说明

```
location /nginx_status {
: # copied from http://blog.kovyrin.net/2006/04/29/monitoring-nginx-with-rrdtool/
: stub_status on;
: access_log   off;
: allow SOME.IP.ADD.RESS;
: deny all;
}
```

指令

- [#stub_status stub_status]

stub_status

语法:*stub_status* on

默认值:*None*

作用域:*location*

创建一个 location 区域启用 stub_status

"stub status" 模块返回的状态信息跟 mathopd's 的状态信息很相似. 返回的状态信息如下：

```
Active connections: 291
server accepts handled requests
: 16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

`active connections` -- 对后端发起的活动连接数

`server accepts handled requests` -- nginx 总共处理了 16630948 个连接, 成功创建 16630948 次握手 (证明中间没有失败的), 总共处理了 31070465 个请求 (平均每次握手处理了 1.8个数据请求)

`reading` -- nginx 读取到客户端的Header信息数

`writing` -- nginx 返回给客户端的Header信息数

`waiting` -- 开启 keep-alive 的情况下，这个值等于 `active - (reading + writing)`，意思就是Nginx说已经处理完正在等候下一次请求指令的驻留连接

Examples

http://blog.kovyrin.net/2006/04/29/monitoring-nginx-with-rrdtool/
http://dev.2xlp.com/svn/nginxconfig/trunk/conf/rrd/__README.txt

HttpSubstitution

This module can search and replace text in the nginx response. It is only available if the

```
--with-http_sub_module option
```

was specified for ./configure.

本模块可以在nginx的回应中查找和替换文本.在编译nginx时必需加上--with-http_sub_module option

例如:

```
location / {
  sub_filter      </head>
  '</head><script language="javascript" src="$script"></script>';
  sub_filter_once on;
}
```

指令

sub_filter

syntax:*sub_filter text substitution*

default:*none*

context:*http, server, location*

sub_filter allows replacing some text in the nginx response with some other text, independently of the source of the data. The matching is case-insensitive. Substitution text may contain variables. Only one substitution rule per location is supported.

sub_filter 允许替换源文件里的多个文本（多次替换）匹配是非常快速的。替换必须包含变量，一个location只能一个替换规则.

sub_filter_once

syntax:*sub_filter_once on|off*

default:*sub_filter_once on*

context:*http, server, location*

sub_filter_once off allows to search and replace all matching lines, the default is replacing only the first one.

sub_filter_once off 允许查找替换所有匹配行，默认只替换第一个.

sub_filter_types

syntax:*sub_filter_types mime-type [mime-type ...]*

default:*sub_filter_types text/html*

context:*http, server, location*

sub_filter_types is used to specify which content types should be checked for *sub_filter*. The default is only *text/html*.

*sub_filter_types*用于指定替换*sub_filter*的类型，默认为*text/html*.

References

Original Documentation

HttpDav模块

这个模块可以为Http webDAV 增加 PUT, DELETE, MKCOL, COPY 和 MOVE 等方法。

这个模块在默认编译的情况下不是被包含的，你需要在编译时指定如下参数：

```
./configure --with-http_dav_module
```

配置范例：

```
location / {
  root     /data/www;
  client_body_temp_path  /data/client_temp;

  dav_methods  PUT DELETE MKCOL COPY MOVE;

  create_full_put_path   on;
  dav_access             group:rw  all:r;

  limit_except  GET {
    allow  192.168.1.0/32;
    deny   all;
  }
}
```

配置指导

dav_access

syntax: dav_access user:permissions [users:permissions] ...

default: dav_access user:rw

context: http, server, location

这个指令是赋予某个目录或文件以访问权，如：

```
dav_access user:rw group:rw all:r;
```

如果给一个用户组或所有用户任何权限，user 这个参数就没有必要设置了，如：

```
dav_access group:rw all:r;
```

dav_methods

syntax: dav_methods [off|put|delete|mkcol|copy|move] ...

default: dav_methods off

context: http, server, location

```
这个指令用于于指定HTTP和WebDAV的方法，设置它为off 时其所有的方法都将无效（忽视你已经设置的方法）。
Put 方法的目标文件必须是和Nginx 的临时文件夹client_body_temp_path 在同一个分区里。
当使用Put 方法创建一个文件时，有可能会通过设定date header 修改文件创建日期。
```

create_full_put_path

syntax: create_full_put_path on|off

default: create_full_put_path off

context: http, server, location

```
默认情况下，Put 方法只能在已存在的目录里创建文件。当然了Nginx 必须得有这个目录的修改和写入权限。
```

GooglePerftools

This module enables Google Performance Tools profiling for workers. This module appeared in nginx version 0.6.29.

By default the module is not built, it is necessary to enable its build with

```
./configure --with-google_perftools_module
```

at compilation time.

Example

```
google_perftools_profiles /path/to/profile;
```

Profiles will be stored as /path/to/profile.

鎸囦护

google_perftools_profiles

syntax:*google_perftools_profiles path*

default:*none*

Directive specifies the base filename of profiles collected by Google Perftools profiler. The workers PIDs will be appended to the specified base filename.

HttpXSLT

This module is a filter which converts an XML response with the aid of one or more XSLT templates.

This module was introduced in 0.7.8 and needs to be enabled via

```
./configure --with-http_xslt_module
```

Example:

```
location / {
  xml_entities       /site/dtd/entities.dtd;
  xslt_stylesheet    /site/xslt/one.xslt   param=value;
  xslt_stylesheet    /site/xslt/two.xslt;

}
```

指令

xslt_entities

syntax:*xml_entities *

default:*no*

context:*http, server, location*

Specifies the DTD file which describes symbolic elements (xml entities). This file is compiled at the stage of configuration. For technical reasons it's not possible to specify entities in the XML being processed, therefore they are ignored, but this specially assigned file is used instead. In this file it is not necessary to describe structure of processed XML, it is sufficient only to declare necessary symbolic elements, for example:

```
  <! ENTITY of nbsp “  “>
```

xslt_stylesheet

syntax:*xslt_stylesheet template [parameter[[parameter... *]] **default:***no*

context:*http, server, location*

Specifies the XSLT template with its parameters. Template is compiled at the stage of configuration. The parameters are assigned as shown:

```
param=value
```

You can specify parameters either one per line, or separate multiple parameters with colon (“: ”) If the parameter itself contains the character “:”, escape it as “%3A”. Furthermore, libxslt requires that string parameters should be quoted by the single or dual quotation marks if they contain non-alphanumeric characters, for example:

```
param1='http%3A//www.example.com': param2=value2
```

It's possible to use variables as parameters, for example, the entire line of the parameters can be substituted with one variable:

```
location / {
  xslt_stylesheet /site/xslt/one.xslt
  $arg_xslt_params

  param1='$value1': param2=value2
  param3=value3;
}
```

It is possible to specify several templates, in which case they would be chained together in the order of their declaration.

xslt_types

syntax:*xslt_types mime-type [mime-type...]*

default:*xslt_types text/xml*

context:*http, server, location*

Permit processing responses with specified MIME-types in addition to “text/xml”. If XSLT output mode is HTML, then the response MIME-type changes to “text/HTML”.

HttpSecureLink

This module computes and checks request URLs for a required security token. This module is not compiled by default and must be specified using the
这个模块计算和检测URL请求中必须的安全标识
这个模块没有默认编译，在编译Nginx时，必须使用明确的配置参数

```
--with-http_secure_link_module
```

argument to configure when compiling Nginx. Note that this module is only supported in nginx version 0.7.18 and higher.
来说明配置.这个模块在Nginx 0.7.18及以上版本中被支持.

Example usage:

```
location /prefix/ {
    secure_link_secret   secret_word;


    if ($secure_link = "") {
        return 403;
    }
}
```

指令

syntax:*secure_link_secret secret_word*

default:*none*

context:*location*

The directive specifies a secret word to verify requests. The full URL for a protected links follows this form:
这个指令明确一个安全关键字来审核请求。完整的被保护的URL如下表：

/**prefix**/*hash*/**reference**

hash is computed as as 使用md5('参考','安全密码')函数计算

```
md5 (reference, secret_word);
```

prefix is the scope of the location block, and must not be '/'. secure_link can only be used in non-root paths.
前缀是定位块的范围，绝对不能使用'/'。安全连接只能被用在非根目录路径。

变量

$secure_link

Automatically set to the reference component of the URL, isolated from the prefix and hash. If the hash is incorrect, an empty string is returned instead.
自动设置模块到URL的关联,分开前缀和HASH。如果HASH不正确，将返回空字符串。

HttpImageFilter

Version: 0.7.54+

This module is a filter for transforming JPEG, GIF and PNG images. It is not enabled by default to enable it, provide this option to ./configure when building nginx:
这个模块用来分发JPEG，GIF和PNG图片。这个没有默认开启，在编译nginx中通过./configure参数配置

```
--with-http_image_filter_module
```

libgd is required to build and run the module. We recommend using the latest version of libgd.
编译和运行这个模块必须安装libgd库。我们推荐使用最新版本的Libgd.

Example Configuration

```
location /img/ {
    proxy_pass     http://backend;
    image_filter   resize  150 100;
    error_page     415   = /empty;

}

location = /empty {
    empty_gif;
}
```

指令

image_filter

Syntax:*image_filter (test|size|resize width height|crop width height)*

Default:*none*

Context:*location*

Specifies the type of transformation to apply to the image, one of the below:
详细的图片后缀类型如下：

- test: checking that the response is indeed an image format JPEG, GIF or PNG. Otherwise, an error 415.
  测试：测试确定图片文件的后缀格式为JPEG，GIF OR PNG。否则返回415错误

- size: Gives information about the image in JSON format. For example,
  尺寸：返回JSON格式的图片信息。例如：

```
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```

Or if an error occurs,
或者如果发生错误,

```
{}
```

- resize: proportionally reduces the image to a specified size.
  调整大小：缩略图片到特定的尺寸

- crop: proportionally reduces the image to a specified size and trims extra edge.
  切割：切割图片到特定的尺寸和特定的分辨率

image_filter_buffer

Syntax:*image_filter_buffer size*

Default:*1M*

Context:*http, server, location*

Sets the maximum size for reading the image.
设置读取文件的最大的尺寸

image_filter_jpeg_quality

Syntax:*image_filter_jpeg_quality [0...100]*

Default:*75*

Context:*http, server, location*

Sets the rate of loss of information when processing the images as **JPEG**. The maximum recommended value is **95**.
设置处理JPEG图片的丢失信息率.推荐的最大值为95

image_filter_transparency

Syntax:*image_filter_transparency on|off*

Default:*on*

Context:*http, server, location*

This directive allows you to disable image transparency in GIF and palette-based PNG to improve image resampling quality.
这个指令容许你关闭GIF图片的透明度和 PNG图片质量

True color PNG alpha-channels are always preserved despite this setting.
真彩色PNG图片保存忽略设定

Note: Grayscale PNG's are untested, but should be handled as truecolor PNGs.
注释：灰度PNG图片未被设置，但应该被作为真彩色PNG图片处理

References

Original Documentation

mail妯″潡

MailCore

Nginx is able to handle and proxy the following mail protocols:
Nginx 能够处理和代理以下邮件协议:

- IMAP
- POP3
- SMTP

认证

nginx uses external HTTP-like server to learn which IMAP/POP backend it should connect to.
nginx的IMAP／POP后端处理类似HTTP服务器的连接方式。

nginx passes authorization information in HTTP headers:
nginx 在HTTP头里传递授权信息:

```
GET /auth HTTP/1.0
Host: auth.server.hostname
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.168.1.1
```

The good response is:
正常的回应是:

```
HTTP/1.0 200 OK      # this line is actually ignored and may not exist at all 这行可能被忽略
Auth-Status: OK
Auth-Server: 192.168.1.10
Auth-Port: 110
Auth-User: newname   # you may override the user name to login to a backend  你可以重写用户
```

When authenticating with APOP for POP3, you must return Auth-Pass as well:
当在POP3中使用APOP协议时，你必须返回验证密码如下:

```
HTTP/1.0 200 OK      # this line is actually ignored and may not exist at all
Auth-Status: OK
Auth-Server: 192.168.1.10
Auth-Port: 110
Auth-User: newname   # you may override the user name to login to a backend
Auth-Pass: password  # this must be the user's password in cleartext 这里必须是明文形式的用户
```

The failed response is:
失败的回应是:

```
HTTP/1.0 200 OK      # this line is actually ignored and may not exist at all
Auth-Status: Invalid login or password
Auth-Wait: 3         # nginx will wait 3 seconds before reading  Nginx在读之前等待3秒
# client's login/passwd again
```

指令

auth

Renamed to pop3_auth in 0.5.15

imap_capabilities

syntax:*imap_capabilities**"capability1" ["capability2" .. "capabilityN"]***

default:*"IMAP4" "IMAP4rev1" "UIDPLUS"*

context:*main, server*

With this directive you can set the list of IMAP protocol extensions presented to the client upon issuing the IMAP command CAPABILITY. STARTTLS is automatically added if you enable the starttls directive.
使用这条指令你可以设置IMAP协议列表来扩展现有的客户端上的IMAP的命令CAPABILITY。如果你启用了STARTTLS那么STARTTLS命令将自动加入。

The current list of standardized IMAP expansions is published on www.iana.org.
现在所列出的标准的IMAP扩展发布在www.iana.org上。

```
mail {
  imap_capabilities NAMESPACE SORT QUOTA;
}
```

Will the defaults be also set, I haven't see this in the source?! (al 2007-05-11)
这个设置了默认了？我在源码中没有看到！

imap_client_buffer

syntax:*imap_client_buffer **size***

default:*4K/8K*

context:*main, server*

With this directive you can set the read buffer for IMAP commands. The default value is equal to the size of a page (this can be either 4K or 8K depending on the platform).

listen

syntax:*listen***address:port [bind]**

default:*no*

context:*server*

The directive specifies the address and port, on which the server accepts requests. It is possible to specify address or port only, besides, an address can be the server name, for example:
当服务器接收请求时，这条指令指定地址和端口。这个可能只知道地址或者端口， 另外，地址可能是服务器名，例如：

```
listen 127.0.0.1:8000;
listen 127.0.0.1;

listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 address(>=0.7.58) are set in square brackets:

```
listen  [::]:8000;
listen  [fe80::1];
```

In directive listen it is possible to indicate the system call bind(2).
listen指令可能显示显示系统命令 bind.

bind -- indicates that it is necessary to make bind(2) separately for this pair of address:port. If several directives listen with identical port but with different addresses and one of the directives listen to all addresses for this port (*:port) then Nginx will make bind(2) only to* :port. In this case the address is determined by the system call getsockname().
bind -- bind指令必须是绑定一对 地址：端口。如果多个指令监听不同地址的相同的端口， 另一个指令监听所有地址的端口(*:端口号)，这样的话Nginx将指执行 bind*的 :端口号 .这样的话地址只能是系统命令getsockname()的值。

pop3_auth

syntax:*pop3_auth***[plain] [apop] [cram-md5]**

default:*plain*

context:*main, server*

With this directive you can set the permitted methods of authentication for POP3 clients:
使用这条指令你可以设置验证POP3客户端的验证方法:

- plain - USER/PASS , AUTH PLAIN , AUTH LOGIN
- apop - APOP
- cram-md5 - AUTH CRAM-MD5

pop3_capabilities

syntax:*pop3_capabilities***"capability1" ["capability2" .. "capabilityN"]***

default:*"TOP" "USER" "UIDL"*

context:*main, server*

With this directive you can set the list of POP3 protocol extensions presented to the client upon issuing the POP3 command CAPA. STLS is automatically added if you enable the starttls directive and SASL is added by the directive auth.
使用这条指令你可以设置 POP3 协议 列表来扩展现有客户端上的POP3命令CAPA。 如果你启用了starttls指令STLS将会自动被添加，通过auth，SASL也将被添加。

protocol

syntax:*protocol***[pop3 | imap | smtp] ;***

default:*IMAP*

context:*server*

This directive set the protocol for this server block.
这条指令设置了服务器块的协议

> server

syntax:*server {...}*

default:*no*

context:*mail*

Directive assigns configuration for the virtual server.
指令指定虚拟服务器的配置

There is no clear separation of the virtual servers ip-based and name-based (the value of the line "Host" header in the request).
这里不区分基于IP的虚拟服务器和基于命名的虚拟服务器（Host指从请求头中获得）

Instead of this by directives listen are described all addresses and ports, on which it is necessary to assume connections for this server, and in directive server_name are indicated all names of servers. Example configurations are described in tuning of virtual servers.
可以替代这个的listen指令描述了连接服务器的所有的地址和端口，和纸条指令的server_name显示了所有的服务器命名。例如 以下配置描述了虚拟服务器的调度

server_name

syntax:*server_name name****fqdn_server_host***

default:*The name of the host, obtained through gethostname()*

context:*mail, server*

Directive assigns the names of virtual server, for example:

```
server {
  server_name   example.com  www.example.com;
}
```

The first name becomes the basic name of server. By default the name of the machine (hostname) is used. It is possible to use "" *for replacing the first part of the name:*
第一个名称被默认为服务器的基本名词.被当作默认的机器名使用,这个也可能被所代替。

```
server {

  server_name   example.com  *.example.com;
}
```

Two of the given name of the above example can be combined into one:
2个给定的名词在以上的例子中被合并成一个

```
server {
  server_name  .example.com;
}
```

The basic name of server is used in an HTTP redirects, if no a "Host" header was in client request or that header does not match any assigned server*name. You can also use just "" to force Nginx to use the "Host" header in the HTTP redirect (note that "" cannot be used as the first name, but you can use a dummy name such as "*" instead):

HTTP重定向中使用了服务器基本名称，如果客户端请求中没有"host"头或者头中没知道server*name.你可以使用"* "来强制Nginx在HTTP重定向中使用"Host"头（注释： "*"不能作为第一命名可以使用""代替）

```
server {
  server_name example.com *;
}
server {
  server_name _ *;
}
```

smtp_auth

syntax:*smtp_auth***[login] [plain] [cram-md5] ;**

default:*login plain*

context:*main, server*

With this directive you can set the permitted methods of authentication for SMTP clients: 使用这个命令你可以设置SMTP客户端的验证方法

- login - AUTH LOGIN
- plain - AUTH PLAIN
- cram-md5 - AUTH CRAM-MD5

smtp_capabilities

syntax:*smtp_capabilities***"capability1" ["capability2" .. "capabilityN"]**

default:*no*

context:*main, server*

With this directive you can set the list of SMTP protocol extensions presented to the client upon issuing the EHLO command. This list is automatically extended by the methods enabled with the directive smtp_auth.
使用这条指令你可以设置SMTP协议扩展现有的客户端通过使用EHLO命令.使用smtp_auth指令这个列表将被自动扩展。

The current list of standardized SMTP expansions is published on www.iana.org . 现在标准的SMTP扩展列表发布在www.iana.org

so_keepalive

syntax:*so_keepalive***on|off**;

default:*off*

context:*main, server*

With this directive you can set the socket SO_KEEPALIVE option for the connection to the IMAP/POP3 backend. In FreeBSD the keepalive option is used for all connections and can be turned off through setsockopt no (see sysctl net.inet.tcp.always_keepalive).
通过使用这条指令你可以设置socket连接IMAP／POP3的SO——KEEPALIVE选项。在FREEBSD中这个 keepalive(保持存活) 选项在所有的连接中都有效并且可以使用setsockopt no来关闭。

timeout

syntax:*timeout****milliseconds;***

default:*60000*

context:*main, server*

With this directive you can set the time out for proxied connections to the back end.
使用这条指令你可以设置 代理连接的超时时间

MailAuth

Example configuration
配置举例：

```
  auth_http           localhost:9000/cgi-bin/nginxauth.cgi;
  auth_http_timeout   5;
}
```

指令

auth_http

syntax:*auth_http****URL***

default:*no*

context:*mail, server*

With this directive you can set the URL to the external HTTP-like server for authorization. A description of the protocol can be found here .
使用这条指令你可以设置URL像HTTP服务器那样的验证。协议描述可以看 这里

auth_http_header

syntax:*auth_http_header****header value***

default:*no*

context:*mail, server*

With this directive you can add a HTTP header and value during the identification process. This makes it possible to use a shared secret to ensure that the request was answered by nginx.
使用这条指令你可以添加HTTP头和值到验证进程中。这个使得可以用共享密码来却不请求得到Nginx的响应

For example:

```
auth_http_header X-NGX-Auth-Key "secret_string";
```

auth_http_timeout

syntax:*auth_http_timeout****milliseconds;***

default:*60000*

context:*mail, server*

With this directive you can set the time out for authentication process.
使用这条指令你可以设置验证进程的超时时间

MailProxy

Nginx can proxy IMAP, POP3, and SMTP protocols.
Ngin可以代理IMAP，POP3和SMTP协议

指令

proxy

syntax:*proxy***on | off**

default:*off*

context:*mail, server*

With this directive you can enable or disable the proxy for mail.
使用这条指令你可以开启和关闭邮件代理

proxy_buffer

syntax:*proxy_buffer***size**

default:*4K/8K*

context:*mail, server*

With this directive you can set the buffer size for the proxy connection. The default value is equal to the size of a page (this can be either 4K or 8K depending on the platform).
使用这条指令你可以设置代理连接的缓存大小。默认值为页面的大小(这个根据平台的不同可能是4K或者8K)

proxy_pass_error_message

syntax:*proxy_pass_error_message***on | off**

default:*off*

context:*mail, server*

With this directive you can pass authentification error messages obtained from the backend back to the client. Usually if authorization in nginx passed successfully then the backend cannot return errors back to the client.
通过这条指令你可以使得客户端忽略从后端获得的验证错误消息.通常验证成不会返回错误消息到客户端。

But for some POP3 servers errors in response to correct password is a regular behavior. For example CommuniGatePro notifies user about overcrowding of the mailbox (or other events) periodically issuing an error in authorization. In this case is worth indicating proxy_error_message on.
但是在POP3服务器中错误是用来更重密码。例如CommuniGatePro 通过验证返回邮箱收件箱已满的错误来通知用户，这样的代理错误信息是很有意义的。

proxy_timeout

syntax:*proxy_timeout***time***

default:*24h*

context:*mail, server*

With this directive you can set the timeout for the proxy connection.
使用这条指令你可以设置代理连接的超时时间。

xclient

syntax:*xclient***on | off***

default:*on*

context:*mail, server*

With this directive you can enable or disable the command XCLIENT with the connection to SMTP backend. This allows the backend to enforce limitations on the client based on IP/HELO/LOGIN.
使用这条指令你可以开启或者关闭命令XCLIENT的SMTP后端连接.这个使得后端强制可以通过IP／HELO／LOGIN限定客户端

If xclient is enabled then nginx first transfers to the backend:
如果xclient被启用，nginx首先转换到后端

```
EHLO server_name
```

Then:

```
XCLIENT PROTO=ESMTP HELO=client_helo ADDR=client_ip LOGIN=authentificated_user NAME=[UNAV
```

MailSSL

This module ensures SSL/TLS support for POP3/IMAP/SMTP. Configuration is practically identical to the configuration of the HTTP SSL module, but checking client certificates is not supported.
这个模块使得POP3／IMAP／SMTP可以使用SSL／TLS.配置已经定义了HTTP SSL模块，但是不支持客户端证书检测。

ssl

syntax:*ssl***on | off***

default:*ssl off*

context:*mail, server*

Enables SSL/TLS for this virtual server.
在虚拟服务器中启用SSL／TLS

ssl_certificate

syntax:*ssl_certificate***file***

default:*cert.pem*

context:*mail, server*

Indicates file with the certificate in PEM format for this virtual server. The same file can contain other certificates, and also secret key in PEM format.
显示虚拟服务器上的PEM格式的证书文件。同一文件可以包含其他的证书和包含PEM格式的安全码。

ssl_certificate_key

syntax:*ssl_certificate_key***file***

default:*cert.pem*

context:*mail, server*

Indicates file with the secret key in PEM format for this virtual server.
显示虚拟服务器中PEM格式的安全码文件

ssl_ciphers

syntax:*ssl_ciphers file***ciphers**

default:*ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP*

context:*mail, server*

Directive describes the permitted ciphers. Ciphers are assigned in the formats supported by OpenSSL.
指令描述了容许的SSL chiphers.chiphers都被使用了OpenSSL支持的格式.

ssl_prefer_server_ciphers

syntax:*ssl_prefer_server_ciphers***on | off**

default:*off*

context:*mail, server*

Requires protocols SSLv3 and TLSv1 server ciphers be preferred over the client's ciphers.
需要SSLv3协议,TLSv1 服务器端米阿么优先于客户端密码

ssl_protocols

syntax:*ssl_protocols***[SSLv2] [SSLv3] [TLSv1]**

default:*SSLv2 SSLv3 TLSv1*

context:*mail, server*

Directive enables the protocols indicated.
指令显示协议

ssl_session_cache

syntax:*ssl_session_cache***[builtin[:size [shared:name:size]**

default:*builtin:20480*

context:*mail, server*

The directive sets the types and sizes of caches to store the SSL sessions.
指令设置了类型和存储SSL 会话的缓存的大小.
The cache types are:
缓存类型为：

- builtin -- the OpenSSL builtin cache, is used inside one worker process only. The cache size is assigned in the number of the sessions.

builtin -- OpenSSL内部缓存,这个只在内部工作进程中被使用.这个缓存大小等同于会话的个数。

- shared -- the cache is shared between all worker processes. The size of cache is assigned in the bytes, 1 MB cache can contain about 4000 sessions. Each shared cache must have arbitrary name. Cache with the same name can be used in several virtual servers.
  shared -- 这个缓存被所有工作进程共享.这个缓存大小用字节标识,1M缓存可以包含大约4000个会话。每个共享缓存都有专有的名称。相同名称的缓存可以被多个服务器使用。

It is possible to use both types of cache simultaneously, for example:
可能同时使用2中类型的缓存，例如

```
ssl_session_cache  builtin:1000  shared:SSL:10m;
```

However, the only shared cache usage without that builtin should be more effective.
然而，共享缓存只有在内部缓存之外使用才能产生更好的效果。

ssl_session_timeout

syntax:*ssl_session_timeout***time***

default:*5m*

context:*mail, server*

Assigns the time during which the client can repeatedly use the parameters of the session, which is stored in the cache.
在使用中客户端重复使用的会话参数被存储在缓存中。

starttls

syntax:*starttls **on | off | only***

default:*off*

context:*mail, server*

- on - permit the use of commands STLS for POP3 and STARTTLS for IMAP/SMTP
  on - 容许在POP3中的STLS命令和IMAP／SMPT中的STARTTLS命令
- off - do not allow command STLS/STARTTLS
  不容许 STLS／STARTTLS命令
- only - announce STLS/STARTTLS support and require that clients use TLS encryption
  only - 宣布支持 STLS／STARTTLS但是需要客户端使用TLS加密

nginx在windows上的安装

写一下简单的安装方法

1.下载软件

需要的软件有nginx,php,mysql(如不需要可不安装)

nginx.org,www.php.net上面有得下

全部解压出来

php基本不用做任何修改（下载直接解压版本的）

2.配置nginx

只需修改一行（nginx/conf/nginx.conf）

```
  fastcgi_param  SCRIPT_FILENAME  d:/nginx/html/$fastcgi_script_name;
```

找到大概这一行修改路径最好是绝对路径（相对路径可以自行探讨）

3.运行软件

先运行php

程序》运行》cmd

到php所在目录运行下面代码

```
php-cgi.exe -b 127.0.0.1:9000 -c php\php.ini
```

再运行nginx

到nginx所在目录

```
nginx.exe -c conf\nginx.conf
```

这样就ok了。

win下可以使用NPMserv更强大

by afen.cn

nginx在freebsd上的安装

一 . 安装必备软件 MySQL+PHP+Pcre

```
cd /usr/ports/database/mysql50-server && make install clean
cd /usr/lang/php5/ && make install clean   选择对cgi mysql等的支持
cd /usr/devel/pcre && make install clean
```

用ports安装 /usr/ports/www/nginx, make install clean

二、弄了一个fastcgi的脚本，来自lighttpd

1) cd /usr/ports/www/lighttpd
2) make
3) cp /usr/ports/www/lighttpd/work/lighttpd-1.4.18/src/spawn-cgi /usr/bin
4) make clean

三、修改配置文件:

1,/usr/local/etc/nginx/nginx.conf:

```
user www www;
worker_processes 10;
error_log /usr/local/etc/nginx/logs/nginx_error.log crit;
worker_rlimit_nofile 51200;
events
{
   use epoll;
   worker_connections 51200;
}
http
{
   include   conf/mime.types;
   default_type application/octet-stream;
   charset gb2312;
   server_names_hash_bucket_size 128;
   keepalive_timeout 60;
   tcp_nodelay on;
   gzip on;
   gzip_min_length 1k;
   gzip_buffers   4 8k;
   gzip_http_version 1.1;
   gzip_types   text/plain application/x-javascript. text/css text/html application/xml;
   server
   {
   listen   80;
   server_name www.test.com;
   index index.html index.htm index.php;
   root /usr/local/www/data/;
   location ~ .*\.php?$
   {
   include fcgi.conf;
   fastcgi_pass 127.0.0.1:9000;
   fastcgi_index index.php;
   }

   log_format access '$remote_addr - $remote_user [$time_local] "$request" '
   '$status $body_bytes_sent "$http_referer" '
   '"$http_user_agent" $http_x_forwarded_for';
   access_log /usr/local/nginx/logs/access.log access;
   }

   }
```

2 ,先将php.ini的配置中 cgi.fix_pathinfo=1 这样php-cgi方能正常使用SCRIPT_FILENAME这个变量。

3,编辑fcgi.conf文件,加入

```
fastcgi_param GATEWAY_INTERFACE CGI/1.1;
fastcgi_param SERVER_SOFTWARE   nginx;

fastcgi_param QUERY_STRING   $query_string;
fastcgi_param REQUEST_METHOD   $request_method;
fastcgi_param CONTENT_TYPE   $content_type;
fastcgi_param CONTENT_LENGTH   $content_length;

fastcgi_param SCRIPT_FILENAME   $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME   $fastcgi_script_name;
fastcgi_param REQUEST_URI   $request_uri;
fastcgi_param DOCUMENT_URI   $document_uri;
fastcgi_param DOCUMENT_ROOT   $document_root;
fastcgi_param SERVER_PROTOCOL   $server_protocol;

fastcgi_param REMOTE_ADDR   $remote_addr;
fastcgi_param REMOTE_PORT   $remote_port;
fastcgi_param SERVER_ADDR   $server_addr;
fastcgi_param SERVER_PORT   $server_port;
fastcgi_param SERVER_NAME   $server_name;
# PHP only, required if PHP was built with --enable-force-cgi-redirect
#fastcgi_param REDIRECT_STATUS 200;
```

四,启动

1,启动fcgi

```
/usr/bin/spawn-fcgi -a 127.0.0.1 -p 9000 -u www -f /usr/local/bin/php-cgi
```

参数说明:

-f 指定调用FastCGI的进程的执行程序位置，根据系统上所装的PHP的情况具体设置
-a 绑定到地址addr
-p 绑定到端口port
-s 绑定到unix socket的路径path
-C 指定产生的FastCGI的进程数，默认为5（仅用于PHP）
-P 指定产生的进程的PID文件路径
-u和-g FastCGI使用什么身份（-u 用户 -g 用户组）运行，Ubuntu下可以使用www-data，其他的根据情况配置，如nobody、apache等

也可建立脚本

1) `ee /usr/bin/php-fastcgi`

```
#!/bin/sh

/usr/bin/spawn-fcgi -a 127.0.0.1 -p 9000 -u www -f /usr/local/bin/php-cgi
```

2) c `hmod 755 /usr/bin/php-fastcgi`

3) `ee /usr/local/etc/rc.d/init-fastcgi`

```
#!/bin/bash

PHP_SCRIPT=/usr/bin/php-fastcgi
RETVAL=0
case "$1" in
start)
$PHP_SCRIPT
RETVAL=$?
;;
stop)
killall -9 php
RETVAL=$?
;;
restart)
killall -9 php
$PHP_SCRIPT
RETVAL=$?
;;
*)
echo "Usage: php-fastcgi {start|stop|restart}"
exit 1
;;
esac
exit $RETVAL
```

4) `chmod 755 /usr/local/etc/rc.d/init-fastcgi`

2,启动nginx

nginx -t -c /usr/local/etc/nginx/nginx.conf 测试配置是否正确

如果屏幕显示以下两行信息，说明配置文件正确：

```
the configuration file /usr/local/etc/nginx/nginx.conf syntax is ok
the configuration file /usr/local/etc/nginx/nginx.conf was tested successfully
```

启动:

```
nginx -c /usr/local/etc/nginx/nginx.conf
```

开机自动启动加入/etc/rc.conf

```
nginx_enable="YES"
```

nginx参数:

-c </path/to/config>为 Nginx 指定一个配置文件，来代替缺省的。

-t不运行，而仅仅测试配置文件。nginx 将检查配置文件的语法的正确性，并尝试打开配置文件中所引用到的文件。

-v显示 nginx 的版本。

-V显示 nginx 的版本，编译器版本和配置参数。

nginx在ubuntu上的安装

1、安装Nginx

```
apt-get install nginx
```

（要最新版本下载下来编译吧）
装完应该能正常运行了。如果之前有装APACHE要改下端口。。。或者直接

```
apt-get remove apache2
/etc/init.d/nginx stop
/etc/init.d/nginx start
```

2、安装php-cgi

```
apt-get install php-cgi
```

（要自定义安装的编译吧）

3、改php-cgi的配置

Ubuntu下是/etc/php5/cgi/php.ini
之前有安装过php的话会复制apache的配置文件
打开cgi.fix_pathinfo选项：

```
cgi.fix_pathinfo=1;
```

4、改Nginx的fastcgi传递参数

Ubuntu下是/etc/nginx/fastcgi_params
默认应该已经设置好了，内容差不多就下面那样：

代码:

```
fastcgi_param QUERY_STRING $query_string;
fastcgi_param REQUEST_METHOD $request_method;
fastcgi_param CONTENT_TYPE $content_type;
fastcgi_param CONTENT_LENGTH $content_length;fastcgi_param SCRIPT_NAME $fastcgi_script_na
fastcgi_param REQUEST_URI $request_uri;
fastcgi_param DOCUMENT_URI $document_uri;
fastcgi_param DOCUMENT_ROOT $document_root;
fastcgi_param SERVER_PROTOCOL $server_protocol;
fastcgi_param GATEWAY_INTERFACE CGI/1.1;
fastcgi_param SERVER_SOFTWARE nginx/$nginx_version;
fastcgi_param REMOTE_ADDR $remote_addr;
fastcgi_param REMOTE_PORT $remote_port;
fastcgi_param SERVER_ADDR $server_addr;
fastcgi_param SERVER_PORT $server_port;
fastcgi_param SERVER_NAME $server_name;
# PHP only, required if PHP was built with –enable-force-cgi-redirect
fastcgi_param REDIRECT_STATUS 200;
```

5、创建VHost配置（这里只说Ubuntu下的，也就路径和包含文件目录的区别了）
在/etc/nginx/sites-available/下创建服务器配置文件比如myserver

```
ln -n /etc/nginx/sites-available/myserver /etc/nginx/sites-enabled/myserver
```

其实懒的话直接创建在enabled里也没问题一。一
内容填：

代码:

```
server {
listen 80;
server_name myserver.com;
access_log /var/log/nginx/myserver.access.log;location / {
root /wwwroot/myserver;
index index.php;
autoindex off;
}
location ~ \.php$ {
include /etc/nginx/fastcgi_params;
fastcgi_param SCRIPT_FILENAME /wwwroot/myserver/$fastcgi_script_name;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
}
# redirect server error pages to the static page /50x.html
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /var/www/nginx-default;
}
# deny access to .htaccess files, if Apache’s document root
location ~ /\.ht {
deny all;
}
}
```

6、安装spawn-fcgi

下载可到这里，这东西已经独立出来了
http://redmine.lighttpd.net/projects/spawn-fcgi/wiki
下完后解压编译

```
sudo ./configure –prefix=/usr/local
sudo make
sudo make install
```

7、启动spaw-fcgi

```
sudo spawn-fcgi -a 127.0.0.1 -p 9000 -u www-data -g www-data -f /usr/bin/php5-cgi -F 10
```

哦，对后面10进程不爽的人可以取消或是加到100

8、重启Nginx

```
/etc/init.d/nginx restart
```

9、定义下hosts就可以看结果啦
编辑/etc/hosts加入

```
127.0.0.1 myserver.com
```

然后打开浏览器就可以看结果啦～
有问题欢迎指出～～

nginx在fedora上的安装

原理：

安装nginx没什么好说的，安装php-cig,让lighthttp的spawn-fcgi对其进行管理

1.用yum仓库安装所需的软件

```
#yum install –y php php-cgi  nginx lighttpd-fastcgi
```

2.生成php-cgi的环境变量配置文件

```
# vim /etc/nginx/fastcgi_params
```

输入以下内容，并把该文件设置为相应属性，可以设置为0777

```
astcgi_param GATEWAY_INTERFACE CGI/1.1;
fastcgi_param SERVER_SOFTWARE nginx;
fastcgi_param QUERY_STRING $query_string;
fastcgi_param REQUEST_METHOD $request_method;
fastcgi_param CONTENT_TYPE $content_type;
fastcgi_param CONTENT_LENGTH $content_length;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name;
fastcgi_param REQUEST_URI $request_uri;
fastcgi_param DOCUMENT_URI $document_uri;
fastcgi_param DOCUMENT_ROOT $document_root;
fastcgi_param SERVER_PROTOCOL $server_protocol;
fastcgi_param REMOTE_ADDR $remote_addr;
fastcgi_param REMOTE_PORT $remote_port;
fastcgi_param SERVER_ADDR $server_addr;
fastcgi_param SERVER_PORT $server_port;
fastcgi_param SERVER_NAME $server_name;
# PHP only, required if PHP was built with –enable-force-cgi-redirect
fastcgi_param REDIRECT_STATUS 200;
```

3.使用spawn-fcgi来控制php-fastcgi的进程

```
# /usr/bin/spawn-fcgi -a 127.0.0.1 -p 9000 -C 200 -u nginx -g nginx -f /usr/bin/php-cgi
```

其中，a绑定的地址，p是端口

-C是PHPCGI的进程数,一般根据自己服务器的内存大小设置，4G服务器，如果是纯WEBSERVER的，可以设置200左右

4.增加server

```
# vim /etc/nginx/nginx.conf
server{
  listen 80;
  server_name pylong.com [www.pylong.com](http://www.pylong.com/);#多个域名，用空格分开
  index index.html index.php index.htm;
  root /var/www/pylong.com;
  location ~ .*.php$
  {
  include /etc/nginx/fastcgi_params;
  fastcgi_pass 127.0.0.1:9000;
  }
  }
```

一些更加详细的conf设置请参看nginx的维基，以及相关资料

一些提示：

yum安装的nginx,相关命令:servie nginx start|stop|reload

把nginx添加到系统启动项里：chkconfig nginx on

其中，a绑定的地址，p是端口

-C是PHPCGI的进程数,一般根据自己服务器的内存大小设置，4G服务器，如果是纯WEBSERVER的，可以设置200左右

4.增加server


```
# vim /etc/nginx/nginx.conf
server{
  listen 80;
  server_name pylong.com [www.pylong.com](http://www.pylong.com/);#多个域名，用空格分开
  index index.html index.php index.htm;
  root /var/www/pylong.com;
  location ~ .*.php$
  {
  include /etc/nginx/fastcgi_params;
  fastcgi_pass 127.0.0.1:9000;
  }
  }
```

一些更加详细的conf设置请参看nginx的维基，以及相关资料

一些提示：

yum安装的nginx,相关命令:servie nginx start|stop|reload

把nginx添加到系统启动项里：chkconfig nginx on

让spawn-fcgi随系统自动启动。

```
vim /etc/rc.loal
```

在文件最后加上

```
# /usr/bin/spawn-fcgi -a 127.0.0.1 -p 9000 -C 200 -u nginx -g nginx -f /usr/bin/php-cgi
```

如果php的session不可用，一般是因为 /var/lib/php/session这个目录的对于nginx不可写，把它所有者改为nginx相应用户已经用户组

转载请注明地址:http://www.pylong.com/?p=6

nginx php-fpm安装配置

nginx本身不能处理PHP，它只是个web服务器，当接收到请求后，如果是php请求，则发给php解释器处理，并把结果返回给客户端。

nginx一般是把请求发fastcgi管理进程处理，fascgi管理进程选择cgi子进程处理结果并返回被nginx

本文以php-fpm为例介绍如何使nginx支持PHP

一、编译安装php-fpm

什么是**PHP-FPM**

PHP-FPM是一个PHP FastCGI管理器，是只用于PHP的,可以在 http://php-fpm.org/download下载得到.

PHP-FPM其实是PHP源代码的一个补丁，旨在将FastCGI进程管理整合进PHP包中。必须将它patch到你的PHP源代码中，在编译安装PHP后才可以使用。

新版**PHP**已经集成**php-fpm**了，不再是第三方的包了，推荐使用。PHP-FPM提供了更好的PHP进程管理方式，可以有效控制内存和进程、可以平滑重载PHP配置，比spawn-fcgi具有更多优点，所以被PHP官方收录了。在./configure的时候带 –enable-fpm参数即可开启PHP-FPM，其它参数都是配置php的，具体选项含义可以查看这里。

安装前准备
centos下执行

```
yum -y install gcc automake autoconf libtool make

yum -y install gcc gcc-c++ glibc

yum -y install libmcrypt-devel mhash-devel libxslt-devel \
libjpeg libjpeg-devel libpng libpng-devel freetype freetype-devel libxml2 libxml2-devel \
zlib zlib-devel glibc glibc-devel glib2 glib2-devel bzip2 bzip2-devel \
ncurses ncurses-devel curl curl-devel e2fsprogs e2fsprogs-devel \
krb5 krb5-devel libidn libidn-devel openssl openssl-devel
```

新版**php-fpm**安装(推荐安装方式)

```
wget http://cn2.php.net/distributions/php-5.4.7.tar.gz
tar zvxf php-5.4.7.tar.gz
cd php-5.4.7
./configure --prefix=/usr/local/php  --enable-fpm --with-mcrypt \
--enable-mbstring --disable-pdo --with-curl --disable-debug  --disable-rpath \
--enable-inline-optimization --with-bz2  --with-zlib --enable-sockets \
--enable-sysvsem --enable-sysvshm --enable-pcntl --enable-mbregex \
--with-mhash --enable-zip --with-pcre-regex --with-mysql --with-mysqli \
--with-gd --with-jpeg-dir

make all install
```

~~旧版手动打补丁**php-fpm**安装（旧版程序已经没有了，大家新版的吧，这里做个展示）~~

~~wget http://cn2.php.net/get/php-5.2.17.tar.gz~~

~~wget http://php-fpm.org/downloads/php-5.2.17-fpm-0.5.14.diff.gz~~

~~tar zvxf php-5.2.17.tar.gz~~

~~gzip -cd php-5.2.17-fpm-0.5.14.diff.gz | patch -d php-5.2.17 -p1~~

~~cd php-5.2.17~~

~~./configure --prefix=/usr/local/php -with-config-file-path=/usr/local/php/etc\~~

~~-with-mysql=/usr/local/mysql\~~

~~-with-mysqli=/usr/local/mysql/bin/mysql_config -with-openssl -enable-fpm -enable-mbstring\~~

~~-with-freetype-dir -with-jpeg-dir -with-png-dir -with-zlib-dir -with-libxml-dir=/usr -enable-xml\~~

~~-with-mhash -with-mcrypt -enable-pcntl -enable-sockets -with-bz2 -with-curl -with-curlwrappers\~~

~~-enable-mbregex -with-gd -enable-gd-native-ttf -enable-zip -enable-soap -with-iconv -enable-bcmath\~~

~~-enable-shmop -enable-sysvsem -enable-inline-optimization -with-ldap -with-ldap-sasl -enable-pdo\~~

~~-with-pdo-mysql~~

~~make all install~~

以上两种方式都可以安装php-fpm，安装后内容放在/usr/local/php目录下

```
root@SNDA-172-17-12-117:/usr/local/src/php-5.4.7# cd /usr/local/php
root@SNDA-172-17-12-117:/usr/local/php# ls
bin  etc  include  lib  php  sbin  var
root@SNDA-172-17-12-117:/usr/local/php#
```

以上就完成了php-fpm的安装。

下面是对php-fpm运行用户进行设置

```
cd /usr/local/php
cp etc/php-fpm.conf.default etc/php-fpm.conf
vi etc/php-fpm.conf
```

修改

```
user = www-data
group = www-data
```

如果www-data用户不存在，那么先添加www-data用户

```
groupadd www-data
useradd -g www-data www-data
```

二、编译安装nginx

然后按照http://www.nginx.cn/install 安装nginx

三、修改nginx配置文件以支持php-fpm

nginx安装完成后，修改nginx配置文件为,nginx.conf

其中server段增加如下配置，注意标红内容配置，否则会出现No input file specified.错误

```
# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
location ~ \.php$ {
root html;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
include fastcgi_params;
}
```

四、创建测试php文件

创建php文件

在/usr/local/nginx/html下创建index.php文件，输入如下内容

```
<?php
echo phpinfo();
?>
```

五、启动服务

启动php-fpm和nginx

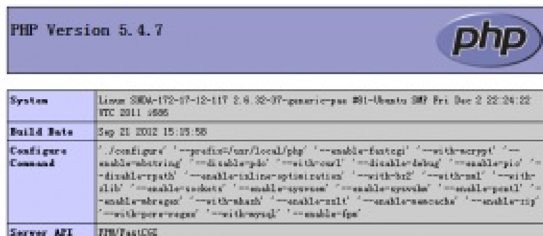
```
/usr/local/php/sbin/php-fpm
#手动打补丁的启动方式/usr/local/php/sbin/php-fpm start

sudo /usr/local/nginx/nginx
```

php-fpm关闭重启见文章结尾

六、浏览器访问

访问http://你的服务器ip/index.php，皆可以见到php信息了。



安装**php-fpm**时可能遇到的错误：

1. **php configure**时出错

configure: error: XML configuration could not be found

```
apt-get install libxml2 libxml2-dev  (ubuntu下)
yum -y install libxml2 libxml2-devel（centos下)
```

1. **Please reinstall the BZip2 distribution**

```
wget http://www.bzip.org/1.0.5/bzip2-1.0.5.tar.gz
tar -zxvf bzip2-1.0.5.tar.gz
cd bzip2-1.0.5
make
make install
```

1. **php**的配置文件中有一行**--with-mysql=/usr**。

安装的时候提示：

```
configure: error: Cannot find MySQL header files under yes.
Note that the MySQL client library is not bundled anymore.
```

这是由于安装mysql时没有安装mysql头文件，或者是路径指定不正确,php找不到mysql的头文件引起的错误提示。

解决方法。

(1.) 查看你的系统有没有安装mysql header

```
find / -name mysql.h
```

如果有。请指定--with-mysql=/跟你的正常路径。

如果没有。请看下一步。

(2.)redhat安装

```
rpm -ivh MySQL-devel-4.1.12-1.i386.rpm
```

(3.)ubuntu安装

```
apt-get install libmysqlclient15-dev
```

(4.)最后一步php的配置选项添加--with-mysql=/usr即可！

4.No input file specified.

```
location ~ \.php$ {
root html;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
include fastcgi_params;
}
```

1. 如果**php configure**时缺库，可以先安装库**(ubuntu**下**)**

```
sudo apt-get install make bison flex gcc patch autoconf subversion locate
sudo apt-get install libxml2-dev libbz2-dev libpcre3-dev libssl-dev zlib1g-dev libmcrypt-
```

1. **mcrypt.h not found. Please reinstall libmcrypt**

```
apt-get install libmcrypt-dev
```

或者

```
cd /usr/local/src
wget http://softlayer.dl.sourceforge.net/sourceforge/mcrypt/libmcrypt-2.5.8.tar.gz
tar -zxvf libmcrypt-2.5.8.tar.gz
cd /usr/local/src/libmcrypt-2.5.8
./configure --prefix=/usr/local
make
make install
```

1. **php-fpm 5.4.7** 如何关闭 重启？

php 5.4.7 下的php-fpm 不再支持 php-fpm 以前具有的 /usr/local/php/sbin/php-fpm (start|stop|reload)等命令，需要使用信号控制：

master进程可以理解以下信号

INT, TERM 立刻终止 QUIT 平滑终止 USR1 重新打开日志文件 USR2 平滑重载所有worker进程并重新载入配置和二进制模块

示例：

php-fpm 关闭：

```
kill -INT `cat /usr/local/php/var/run/php-fpm.pid`
```

php-fpm 重启：

```
kill -USR2 `cat /usr/local/php/var/run/php-fpm.pid`
```

查看php-fpm进程数：

```
ps aux | grep -c php-fpm
```

8.命令行下执行php，提示找不到命令

```
-bash: /usr/bin/php: No such file or directory

vi /etc/profile
```

在文件底部增加一行配置

```
export PATH=/usr/local/php/bin:$PATH
```

保存退出

```
source /etc/profile
```

配置示例和方法

完整例子

两个虚拟主机(纯静态-html 支持) - Two Virtual Hosts, Serving Static Files

```
http {
: server {
: listen          80;
: server_name     www.domain1.com;
: access_log      logs/domain1.access.log main;
: location / {
: index index.html;
: root  /var/www/domain1.com/htdocs;
: }
: }
: server {
: listen          80;
: server_name     www.domain2.com;
: access_log      logs/domain2.access.log main;
: location / {
: index index.html;
: root  /var/www/domain2.com/htdocs;
: }
: }
}
```

虚拟主机标准配置(简化) - A Default Catchall Virtual Host

```
http {
: server {
: listen          80 default;
: server_name     _ *;
: access_log      logs/default.access.log main;
: location / {
: index index.html;
: root  /var/www/default/htdocs;
: }
: }
}
```

在父文件夹中建立子文件夹以指向子域名 - Wildcard Subdomains in a Parent Folder

这是一个添加子域名(或是当DNS已指向服务器时添加一个新域名)的简单方法。需要注意的是，我已经将FCGI配置进该文件了。如果你只想使服务器为静态文件服务，可以直接将FCGI配置信息注释掉，然后将默认主页文件变成index.html。

这个简单的方法比起为每一个域名建立一个 vhost.conf 配置文件来讲，只需要在现有的配置文件中增加如下内容：

This is just a really easy way to keep adding new subdomains, or to add new domains automatically when DNS records are pointed at the server. Note that I have included FCGI here as well. If you want to just serve static files, strip out the FCGI config and change the default document to index.html. Rather than creating a new vhost.conf file for every domain, just create one of these:

```
server {
: # Replace this port with the right one for your requirements
: # 根据你的需求改变此端口
: listen       80;  #could also be 1.2.3.4:80 也可以是1.2.3.4:80的形式
: # Multiple hostnames seperated by spaces.  Replace these as well.
: # 多个主机名可以用空格隔开，当然这个信息也是需要按照你的需求而改变的。
: server_name  star.yourdomain.com *.yourdomain.com www.*.yourdomain.com;
: #Alternately: _ *
: #或者可以使用：_ * (具体内容参见本维基其他页面)
: root /PATH/TO/WEBROOT/$host;
: error_page  404              http://yourdomain.com/errors/404.html;
: access_log  logs/star.yourdomain.com.access.log;
: location / {
: root   /PATH/TO/WEBROOT/$host/;
: index  index.php;
: }
: # serve static files directly
: # 直接支持静态文件 (从配置上看来不是直接支持啊)
: location ~* ^.+.(jpg|jpeg|gif|css|png|js|ico|html)$ {
: access_log        off;
: expires           30d;
: }
: location ~ .php$ {
: # By all means use a different server for the fcgi processes if you need to
: # 如果需要，你可以为不同的FCGI进程设置不同的服务信息
: fastcgi_pass   127.0.0.1:YOURFCGIPORTHERE;
: fastcgi_index  index.php;
: fastcgi_param  SCRIPT_FILENAME  /PATH/TO/WEBROOT/$host/$fastcgi_script_name;
: fastcgi_param  QUERY_STRING     $query_string;
: fastcgi_param  REQUEST_METHOD   $request_method;
: fastcgi_param  CONTENT_TYPE     $content_type;
: fastcgi_param  CONTENT_LENGTH   $content_length;
: fastcgi_intercept_errors on;
: }
: location ~ /\.ht {
: deny  all;
: }
: }
```

完整例子2

这是来自 Nginx官方网站 的一个例子。

```
#!nginx
: # 使用的用户和组
: user  www www;
: # 指定工作衍生进程数
: worker_processes  2;
: # 指定 pid 存放的路径
: pid /var/run/nginx.pid;

: # [ debug | info | notice | warn | error | crit ]
: # 可以在下方直接使用 [ debug | info | notice | warn | error | crit ]  参数
: error_log  /var/log/nginx.error_log  info;

: events {
: # 允许的连接数
: connections   2000;
: # use [ kqueue | rtsig | epoll | /dev/poll | select | poll ] ;
: # 具体内容查看 http://wiki.codemongers.com/事件模型
: use kqueue;
: }

: http {
: include       conf/mime.types;
: default_type  application/octet-stream;

: log_format main      '$remote_addr - $remote_user [$time_local]  '
: '"$request" $status $bytes_sent '
: '"$http_referer" "$http_user_agent" '
: '"$gzip_ratio"';

: log_format download  '$remote_addr - $remote_user [$time_local]  '
: '"$request" $status $bytes_sent '
: '"$http_referer" "$http_user_agent" '
: '"$http_range" "$sent_http_content_range"';

: client_header_timeout  3m;
: client_body_timeout    3m;
: send_timeout           3m;

: client_header_buffer_size    1k;
: large_client_header_buffers  4 4k;

: gzip on;
: gzip_min_length  1100;
: gzip_buffers     4 8k;
: gzip_types       text/plain;

: output_buffers   1 32k;
: postpone_output  1460;

: sendfile         on;
: tcp_nopush       on;
: tcp_nodelay      on;
: send_lowat       12000;

: keepalive_timeout  75 20;

: #lingering_time     30;
: #lingering_timeout  10;
: #reset_timedout_connection  on;


: server {
: listen        one.example.com;
```

```
: server_name   one.example.com  www.one.example.com;

: access_log   /var/log/nginx.access_log  main;

: location / {
: proxy_pass         http://127.0.0.1/;
: proxy_redirect     off;

: proxy_set_header   Host             $host;
: proxy_set_header   X-Real-IP        $remote_addr;
: #proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;

: client_max_body_size       10m;
: client_body_buffer_size    128k;

: client_body_temp_path      /var/nginx/client_body_temp;

: proxy_connect_timeout      90;
: proxy_send_timeout         90;
: proxy_read_timeout         90;
: proxy_send_lowat           12000;

: proxy_buffer_size          4k;
: proxy_buffers              4 32k;
: proxy_busy_buffers_size    64k;
: proxy_temp_file_write_size 64k;

: proxy_temp_path            /var/nginx/proxy_temp;

: charset  koi8-r;
: }

: error_page  404  /404.html;

: location /404.html {
: root  /spool/www;

: charset         on;
: source_charset  koi8-r;
: }

: location /old_stuff/ {
: rewrite   ^/old_stuff/(.*)$  /new_stuff/$1  permanent;
: }

: location /download/ {

: valid_referers  none  blocked  server_names  *.example.com;

: if ($invalid_referer) {
: #rewrite   ^/   http://www.example.com/;
: return   403;
: }

: #rewrite_log  on;

: # rewrite /download/*/mp3/*.any_ext to /download/*/mp3/*.mp3
: rewrite ^/(download/.*)/mp3/(.*)\..*$
: /$1/mp3/$2.mp3                   break;

: root         /spool/www;
: #autoindex    on;
: access_log   /var/log/nginx-download.access_log  download;
: }

: location ~* ^.+\.(jpg|jpeg|gif)$ {
: root         /spool/www;
: access_log   off;
: expires      30d;
: }
: }
: }
```

虚拟主机

两个虚拟主机(纯静态-html 支持) - Two Virtual Hosts, Serving Static Files

```
http {
: server {
: listen          80;
: server_name     www.domain1.com;
: access_log      logs/domain1.access.log main;
: location / {
: index index.html;
: root  /var/www/domain1.com/htdocs;
: }
: }
: server {
: listen          80;
: server_name     www.domain2.com;
: access_log      logs/domain2.access.log main;
: location / {
: index index.html;
: root  /var/www/domain2.com/htdocs;
: }
: }
}
```

虚拟主机标准配置(简化) - A Default Catchall Virtual Host

```
http {
: server {
: listen          80 default;
: server_name     _ *;
: access_log      logs/default.access.log main;
: location / {
: index index.html;
: root  /var/www/default/htdocs;
: }
: }
}
```

在父文件夹中建立子文件夹以指向子域名 - Wildcard Subdomains in a Parent Folder

这是一个添加子域名(或是当DNS已指向服务器时添加一个新域名)的简单方法。需要注意的是，我已经将FCGI配置进该文件了。如果你只想使服务器为静态文件服务，可以直接将FCGI配置信息注释掉，然后将默认主页文件变成index.html。

这个简单的方法比起为每一个域名建立一个 vhost.conf 配置文件来讲，只需要在现有的配置文件中增加如下内容：

This is just a really easy way to keep adding new subdomains, or to add new domains automatically when DNS records are pointed at the server. Note that I have included FCGI here as well. If you want to just serve static files, strip out the FCGI config and change the default document to index.html. Rather than creating a new vhost.conf file for every domain, just create one of these:

```
server {
: # Replace this port with the right one for your requirements
: # 根据你的需求改变此端口
: listen       80;  #could also be 1.2.3.4:80 也可以是1.2.3.4:80的形式
: # Multiple hostnames seperated by spaces.  Replace these as well.
: # 多个主机名可以用空格隔开，当然这个信息也是需要按照你的需求而改变的。
: server_name  star.yourdomain.com *.yourdomain.com www.*.yourdomain.com;
: #Alternately: _ *
: #或者可以使用：_ * (具体内容参见本维基其他页面)
: root /PATH/TO/WEBROOT/$host;
: error_page  404              http://yourdomain.com/errors/404.html;
: access_log  logs/star.yourdomain.com.access.log;
: location / {
: root   /PATH/TO/WEBROOT/$host/;
: index  index.php;
: }
: # serve static files directly
: # 直接支持静态文件 (从配置上看来不是直接支持)
: location ~* ^.+.(jpg|jpeg|gif|css|png|js|ico|html)$ {
: access_log        off;
: expires           30d;
: }
: location ~ .php$ {
: # By all means use a different server for the fcgi processes if you need to
: # 如果需要，你可以为不同的FCGI进程设置不同的服务信息
: fastcgi_pass   127.0.0.1:YOURFCGIPORTHERE;
: fastcgi_index  index.php;
: fastcgi_param  SCRIPT_FILENAME  /PATH/TO/WEBROOT/$host/$fastcgi_script_name;
: fastcgi_param  QUERY_STRING     $query_string;
: fastcgi_param  REQUEST_METHOD   $request_method;
: fastcgi_param  CONTENT_TYPE     $content_type;
: fastcgi_param  CONTENT_LENGTH   $content_length;
: fastcgi_intercept_errors on;
: }
: location ~ /\.ht {
: deny  all;
: }
: }
```

负载均衡

一个简单的负载均衡的示例，把**www.domain.com**均衡到本机不同的端口，也可以改为均衡到不同的地址上。**>**

```
http {
: upstream myproject {
: server 127.0.0.1:8000 weight=3;
: server 127.0.0.1:8001;
: server 127.0.0.1:8002;
: server 127.0.0.1:8003;
: }

: server {
: listen 80;
: server_name www.domain.com;
: location / {
: proxy_pass http://myproject;
: }
: }
}
```

nginx防盗链

```
location ~* \.(gif|jpg|png|swf|flv)$ {
root html
valid_referers none blocked *.nginxcn.com;
if ($invalid_referer) {
rewrite ^/ [www.nginx.cn](http://www.nginx.cn/)
#return 404;
}
}
```

前面的root可以不要如果你在server{}中有设置可以不需要设定

HWLoadbalancerCheckErrors

Some Hardware Load-balancers such Cisco's CSS and BigIP Products test the readiness of the backend Machines with **SYN-ACK-RST**.

This behavior causes a 400 error in nginx.

With the GEO Module and the **if-Statement** you can omit these entries:

```
http {

  geo  $lb  {
    default      0;
    10.1.1.1/32  1;   # LB IPs

    10.1.1.2/32  1;
  }

  # ...

  server {

    # ...
    access_log   /path/to/log;
    error_page 400 /400;

    location = /400 {

      if ($lb) {
        access_log  off;
      }
      return 400;
    }

  }
}
```